DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Y62

A VIRTUAL
STATISTICAL MECHANICAL NEURAL COMPUTER

by

Charles P. Yost

December 1987

Thesis Advisor: Lester Ingber

Approved for public release; distribution is unlimited.

T239338

URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
DECLASSIFICATION / DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (*If applicable*) 61
7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000
NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (*If applicable*)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO

TITLE (*Include Security Classification*)

A VIRTUAL STATISTICAL MECHANICAL NEURAL COMPUTER

PERSONAL AUTHOR(S)

YOST, Charles P.

TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987, DECEMBER	92

SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Statistical Mechanics, Neural computers, nonlinear probability distributions

ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This study applies recently developed statistical algorithms and an innovative large system scaling technique to the problem of implementing a virtual neural computer. Once the techniques are shown to faithfully model highly nonlinear, nonequilibrium probability distributions, they are applied to the brain.

The statistical mechanical neural computer (SMNC) developed in this thesis makes use of scaling to effectively filter the information flow and to model its contents. The implications for command and control are the SMNC's ability to recognize patterns of previously stored information detecting similarities between new and old data. The purpose of the SMNC is to serve as a decision aid that will contain high quality information about specific nonlinear relationships related to system variables, through the aggregation of information into coarse-grained data at a mesoscopic level. This should

DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
. NAME OF RESPONSIBLE INDIVIDUAL Lester Ingber	22b TELEPHONE (*Include Area Code*) 408-646-2687
22c. OFFICE SYMBOL 61Ii	

FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986 606-243

19. Abstract (cont).

give the battle field commander or the Wall Street analyst, the ability to more accurately forecast the most likely course of events a given scenario would follow based on its recent history.

The SMNC may be applied to the study of any stochastic processes. Its methods for the aggregation and scaling of data make it an effective tool for the study of both short and long term behavior in nonlinear nonequilibrium systems.

Approved for public release; distribution is unlimited.

A Virtual Statistical Mechanical Neural Computer

by

Charles P. Yost
Lieutenant Commander, United States Navy
B.S., Miami University, 1975
M.B.A., University of West Florida, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

December 1987

Thesis
Y62
c.1

ABSTRACT

This study applies recently developed statistical algorithms and an innovative large system scaling technique to the problem of implementing a virtual neural computer. Once the techniques are shown to faithfully model highly nonlinear, nonequilibrium probability distributions, they are applied to the brain.

The statistical mechanical neural computer (SMNC) developed in this thesis makes use of scaling to effectively filter the information flow and to model its contents. The implications for command and control are the SMNC's ability to recognize patterns of previously stored information detecting similarities between new and old data. The purpose of the SMNC is to serve as a decision aid that will contain high quality information about specific nonlinear relationships related to system variables, through the aggregation of information into coarse-grained data at a mesoscopic level. This should give the user, be it battlefield commander or Wall Street analyst, the ability to more accurately forecast the most likely course of events a given scenario would follow based on its recent history.

The SMNC is well suited for the study of stochastic processes. Its methods for the aggregation and scaling of data make it an effective tool for the study of both short and long term behavior in nonlinear nonequilibrium systems.

TABLE OF CONTENTS

I. INTRODUCTION 7

II. PAST AND PRESENT TECHNOLOGIES 10

A. BACKGROUND: THE BRAIN 10

B. EARLY ATTEMPTS AT MODELING THE BRAIN 12

C. CURRENT APPROACHES 13

1. Optical Neural Computers 14

2. Parallelism 15

3. Other Networking Concepts 16

III. A STATISTICAL MECHANICAL NEURAL COMPUTER 18

A. INTRODUCTION 18

B. DOMAINS 19

1. The Microscopic Domain 19

2. The Mesoscopic Domain 22

C. CONCLUSIONS 28

IV. OPERATION OF THE SMNC 30

A. INTRODUCTION 30

1. The Microscopic Scale 31

2. The Mesoscopic Scale 32

B. COMPUTATIONAL EFFICIENCY 32

C. VERIFICATION 35

V. CONCLUSIONS 45

A. INTRODUCTION 45

B. SUMMATION OF RESULTS 46

C. VARIATION OF PARAMETERS 50

D. RECOMMENDATIONS 52

APPENDIX A: SOURCE CODE FOR THE NEOCORTICAL SMNC 54

1. THE MISCELLANEOUS MODULE 54

a. Data Structures and the SMNC 55

b. Random Number Routines 56

2. THE INITIALIZATION MODULE 57

a. The Mesoscopic Scale 58

b. The Microscopic Scale 58

3. THE UPDATE MODULE 59

a. The Microscopic Update Cycle 59

b. The Mesoscopic Update Cycle 60

APPENDIX B: SOURCE CODE FOR THE MESOSCOPIC SMNC 75

APPENDIX C: MONTE CARLO CALCULATIONS 82

LIST OF REFERENCES 85

INITIAL DISTRIBUTION LIST 88

I. INTRODUCTION

It's a familiar scene, whether you are a battlefield commander, a corporate executive or a Wall Street analyst; the facts keep changing faster than your ability to comprehend them. You and your assets are stretched to the limit as the risks involved escalate. Your decisions are crucial to the success of your mission, reputation, or business and time is running out. How do you best use the information you have? What is the best decision? What path will the events likely follow? How can you use the information you have to optimize your decisions and what influence will these decisions have on the actions that follow? These are questions we have all asked ourselves at one time or another. If only the crystal ball were a bit clearer or our intuition a bit better.

Today's computers cannot match the real-time performance of the human retina. It is estimated that to simulate the computational powers of the human eye would take a minimum of 100 years on a Cray supercomputer [1]. The human brain is one of nature's most complicated works of art. Its neocortex is many-folded and more complex than the retina. As a computer it encorporates parallelism on a scale far beyond anything man has yet to devise. It is capable of complex pattern recognition, an ability that the most modern computers can match only on the smallest of scales. Yet the brain is terribly inefficient and easily outperformed by the smallest calculator in dealing with simple linear operations. Although the brain cannot match the speed and accuracy of a computer, for some things it more than compensates for this through its ability for abstraction and reasoning. It is the ability to recall past events that enables the brain to respond to changing situations. This has also allowed man to learn from his environment

by associating similar patterns with similar results, that is, to recognize danger in a situation not previously experienced.

This thesis represents an effort to incorporate such "intuition" into complex multivariate nonlinear command, control and communications ($C^{3)}$ systems requiring stochastic or probabilistic treatment. In C^{3}, as in many other systems ranging from neuroscience to nuclear physics, data rates often exceed that of human comprehension. Current solutions include the construction of networks of many units computing simultaneously in a manner similar to the way neurons cooperate in the nervous systems of living organisms [2, 3]. Unfortunately the huge connection matrix required to account for all interneural connections has made this approach impossible in the past and limits its practicality in the present.

Through the use of statistical mechanical techniques to model neocortical interactions [4], a statistical mechanical neural computer (SMNC) incorporates a mesoscopic scaling level that enables a timely yet robust means of handling large quantities of data. It is the existence of several scales of neocortical interactions that suggest the use of nonlinear nonequilibrium statistical mechanics. Through coarse-graining, the model is capable of explaining macroscopic neocortical activity while retaining an accurate average description of the underlying microscopic synaptic activity. This purpose of this study was to establish an ordered 2-dimensional mesoscopic substrate upon which a macroscopic formulation of statistical firings could be developed, and to show that through the use of a control structure at the microscopic level, the validity of the scaling can be proven.

The SMNC which this thesis proposes will provide the battlefield commander, the logistics planner or personnel coordinator with a decision making tool for discerning the best course of action based on uncertain, incomplete, or contradictory data through approaches the modern computer is yet incapable of making. The code and memory requirements to accomplish this feat are such that the program can be run on a personal computer, in the field if necessary.

Chapter II provides the reader with an overview of current approaches to the topic of parallel processing and neural nets, as well as further motivation for this paper. A Statistical Mechanical Neural Computer, Chapter III, describes the algorithms and stochastic statistical mechanics that apply to the neocortical SMNC [5]. In Chapter IV the operation of the mesoscopic statistical mechanical neural computer is discussed and the steps taken for the verification of the code used with the mesoscopic scale in 1-dimension are outlined. Chapter V, the conclusion, discusses the results thus far obtained from the SMNC and recommendation for further research and application. Appendix A contains a partial listing of the source code for the microscopic scale as well as a description of the associated data structures, algorithms and mathematics used to model the brain. Appendix B contains a description of the code used for the 1-dimensional mesoscopic computer. Appendix C discusses the Cauchy-driven Monte-Carlo methods used for the approximation of nonlinear nonequilibrium events.

II. PAST AND PRESENT TECHNOLOGIES

This chapter looks at several of the early attempts to model the brain as well as current approaches to neural networks and other methods being applied to the problems of information flow and data analysis in large scale systems. The brain is one of nature's most complicated systems consisting of highly specialized cells with unique structure and purpose, yet it provides scientists an excellent example of computer architecture to model and emulate. By modeling the neurons of the brain, scientists hope to gain the insight and vision needed for the next generation of faster and more powerful computers.

A. BACKGROUND: THE BRAIN

The basic operating unit of the brain is the neuron, depicted in Figure 2.1 [6]. A typical neuron consists of a cell body, ranging from about 5 to 100 micrometers in diameter. Emanating from the cell body is one major fiber called an axon, and a number of fibrous branches called dendrites [7, 8]. The dendrites receive incoming signals and send them to the cell body for integration and further propagation. The human brain contains about 10^{10} neurons, each capable of sending information to and receiving information from about 10^4 of its neighbors, mostly nearest-neighbors. Each neuron can also communicate with a small fraction of other more distant neurons. Stimuli affecting a neuron are translated into all or nothing depolarization pulses, or action potentials, which are then propagated along the axon. Enough depolarization (10 - 20 mV) within a period of 5-10 msec may cause the neuron to fire by generating its own electrical pulse or

action potential, thereby further adding to the total intensity of the initial action potential [9].

Current technology has enabled the modeling of the actual properties of many biological and physiological nonlinear nonequilibrium systems through the application of statistical mechanics. The physiological properties of the neural system within the brain can be approximated through the application of a nonlinear dynamic assemblage of quasi-random decision elements, also known as a neural network. A dynamic neural network can be characterized by a complex pattern of variable neuron to neuron connections, with the variability being determined stochastically [10].

Recent studies show that several levels of scaling of neocortical interactions exist. It is the existence of these scales of interactions that suggests the use of nonlinear nonequilibrium statistical mechanics. Through coarse-graining, a method of treating nonlinear nonequilibrium statistical systems, a model is capable of explaining

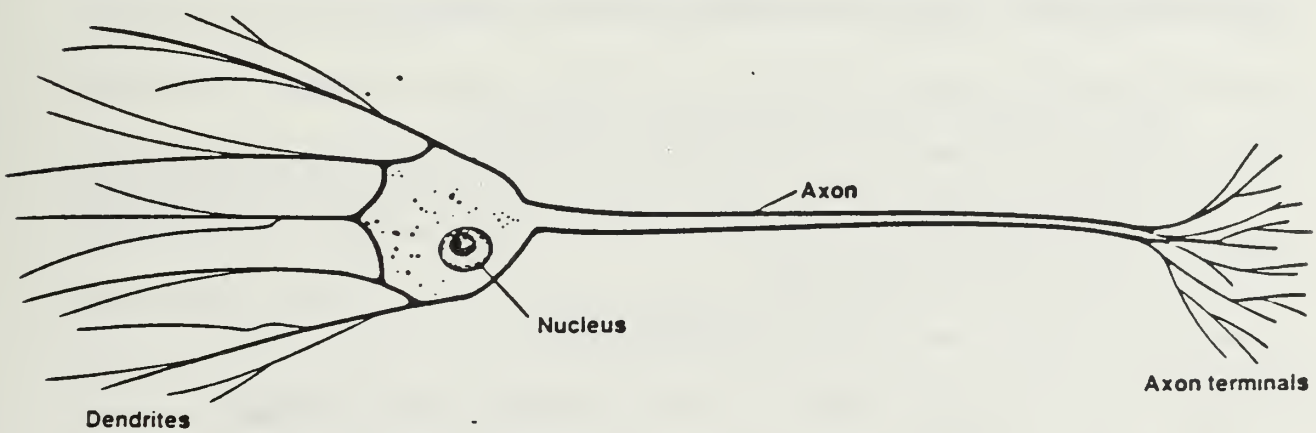

Figure 2.1 A Typical Neuron

macroscopic neocortical activity while retaining an accurate description of the underlying microscopic synaptic activity. As found in most nonlinear systems, a mesoscopic or intermediate scale is also required to accommodate a statistical model of the microscopic scale. The mesoscopic scale can be defined as the spatial extent of a minicolumn. A minicolumn is comprised of clusters of about 110 neurons. A large majority of these have interactions with their nearest neighbors (10^3 other minicolumns within about 1 mm) and thus provide a physical context for a macroscopic region [4].

Each minicolumn has two basic types of neurons: excitatory (E) and inhibitory (I). We assume that there are approximately 80 excitatory neurons and 30 inhibitory neurons in a minicolumn. At any given time, each neuron within this network of interconnected neurons is either firing or not firing. The decision whether to fire or not is stochastically determined and depends on the strength of existing stimuli reaching the neuron and the existing background noise induced from synaptic interactions with other neurons. A neuron *may* fire anytime its threshold is exceeded. Firing is an all or nothing neural response based on an integrate and fire at threshold scheme [11].

B. EARLY ATTEMPTS AT MODELING THE BRAIN

As early as 1943 the concept of a neural net was presented by McCullock and Pitts [12]. They assumed the following:

- the activity of the neuron was an all or nothing process,
- a fixed number of synapses needed to be stimulated prior to neuron excitation,
- excitation was independent of previous activity,
- delays associated with the system involved only synaptic delays,

- an inhibitory response exists that is capable of countering, and
- the excitation of the neuron and its entire structure is time invariant.

Rosenblatt [12] carried the ideas of McCullock and Pitts further with his perceptron concept. The theory proposed by Rosenblatt dealt with an entire class of brain models and introduced the term perceptron. Rosenblatt defined perceptron as a network of sensory, association, and response units with a variable interaction matrix of coupling coefficients for joining all pairs of sensory units relying on the sequence of past activity states of the network. Rosenblatt sought a physical system capable of perceiving its environment, and learning to recognize events encountered in the past. He concluded that his perceptron model was capable of learning to duplicate the performance of any finite task. In 1960, Rosenblatt demonstrated that a 20 X 20 network of perceptrons, implemented in hardware, was capable of recognizing the letters of the alphabet.

The proposals of Rosenblatt were not fully accepted by others. While codirectors of the Artificial Intelligence Group at MIT, Minsky and Papert [13] published a book, *Perceptrons,* which was less than supportive on the topic. They concluded that the idea of a parallel computer modeled after the brain was ahead of its time and that much more research into this topic was needed before an accurate model could be built. The effect this had on the field of neural computers was not favorable, and it was not until the early 70's that interest in neural computers resumed.

C. CURRENT APPROACHES

Ongoing research and applications of our knowledge about the brain to computers can be divided into several broad categories. One large body of research deals with neural computers and their implementation through optical means [14-16]. Yet

another approach views the neural computer as a device employing parallelism on a massive scale [17]. Some promote the virtues of a neural computer through network theory [3, 18, 19].

1. Optical Neural Computers

Conventional computers are comprised of electronic switching units with some small degree of interconnectivity. Processing of information is generally done in a linear step-wise manner. The brain, on the other hand, employs a large number of interconnected neurons, each capable of communicating simultaneously with a vast number of its neighbors. Neural net systems model the human brain using the neuron as the basic unit. Some neural computers are being designed to solve problems through optical elements arranged the same as neurons are arranged in the brain. Current applications include the one proposed by Psaltis of Cal Tech and Farhart of the University of Pennsylvania, working for DARPA on the optical implementation of a neural network computer using an array of light emitting diodes which represent logic units with binary states. Nonlinear feedback is achieved by the use of an optical vector matrix multiplier and then through a threshold circuit to another array of light emitting diodes. Each output LED assesses the state of its input and fires according to whether or not its threshold has been exceeded [15]. Mostafa and Psaltis [15] suggest that the anatomical structure of the brain serves as an organizational principle by which associations can be readily established between what is stored in memory and input data. It is their assessment that optical technology fits well with the concept of a neural computer due to the technology's strengths, that is, a large number of interconnections and processing elements working simultaneously on one or many problems.

2. Parallelism

Another approach to the problem of how to rapidly handle large amounts of data is through the use of massive parallelism. This is much the way the brain handles incoming sensory information and is the reason this approach is viewed by many as neural networking. Instead of waiting for the information to be collected in total, the brain begins processing the information as it becomes available and does so by distributing data to various segments to co-process, or deal with in parallel. The neural substrate of memory and learning is a question of great importance. The success of parallelism in computing has been suggested to be related to the fact that human intelligence has evolved along the lines of massively parallel hardware [20].

W. D. Hillis has designed a computer system referred to as the Connection Machine [17]. Incorporated in the Connection Machine is "data level parallelism", which refers to a strategy in computer design that strives to fit computer architecture to the problem by using inherent parallelism. "Data level parallelism" is appropriate for tasks dealing with large numbers of independent data elements capable of manipulation in parallel by multiple processors. The Connection Machine uses a network of 65,536 individual single bit processors, each with 4096 bits of memory. Instructions are broadcast to all the processors which execute in parallel. A massive interconnection system, or router, connects the processors to permit any processor to communicate with any other processor. Several applications for the Connection Machine's architecture have been suggested including document retrieval, fluid dynamics, and Strategic Defense Initiative (SDI) [2, 17]. Key concepts include massively parallel processing, high speed paging and cluster analysis.

Other users view neural networks as a form of natural intelligence or NI (as opposed to AI). Defining AI as a Turing like software development (implemented by Von Neuman type hardware with the noted exception of the Connection Machine), Harold Szu of NRL, views NI as a global connection machine implemented through a combination of algorithms and architectures for both efficient interface and computational rates [21].

Hecht-Nielsen [3] states that an artificial neural system is the engineering discipline concerning the design, implementation, and application of dynamic systems capable of processing information by means of response to continuous input. His stated goal is the creation of a man-made system that is capable of processing information the same as the brain by allowing a network of neural units to adjust to their environment. The properties of associative memory have been adopted by Hecht-Nielsen Neurocomputer Corporation. This company has also released a neural network description language language called AXON which facilitates the description of any type of neural network architecture. Finially, there has recently been formed an International Neural Network Society, whose purpose is to create a scientific and educational forum for students, scientists, and engineers to learn about and advance the state of knowledge in this field.

Other companies have also marketed parallel systems, including Intel, INMOS, and Floating Point Systems [11].

3. Other Networking Concepts

Hopfield and Tank of Cal Tech [16] have developed a neural network computer that was applied to the historic traveling salesman problem. Connections

between the neural units served to model the distances between cities. Using 10 cities their results were consistently among the first or second best when compared to a main frame computer. When 30 cities were used for 10^{30} possible choices, the neural computer's answers were among the best 100 million, which were all within 10^{-21} of each other. This was done in less than 0.1 second which compares to over an hour on a large dedicated main frame computer.

Hopfield and Tank [18] also propose the use of circuits of nonlinear graded-response neural units organized into networks of symmetric synaptic connections to prove associative learning. They apply the computational properties of biological organisms to the field of computer design. Their goal is to model a neuron's effective input, output and internal state, as well as the relation between its input and output. Their model fails to take into account that associations are often asymmetric. The symmetry of their model is not necessary to prove associative learning and memory storage by neural like networks [15].

III. A STATISTICAL MECHANICAL NEURAL COMPUTER

This chapter addresses the data structures and design decisions behind a statistical mechanical neural computer. It also introduces some of the theory and algorithms that enable such a system to not only model the brain, but also model Red and Blue forces in combat, or model the hostile actions of an enemy while embedded in an SDI satellite.

A. INTRODUCTION

Ingber [4, 5, 22-25] has studied the dynamics of the brain and discusses his results in a series of papers on the statistical mechanics of neocortical interactions by (SMNI). His conclusions serve as the initial development of the SMNC by providing a means to test and validate data on several scales. This data can be aggregated to yield information on a mesoscopic scale about variables such as measures of force or measures of effectiveness. At the same time, calculations made at a microscopic level will enhance decision making at the command, or macroscopic level. The microscopic level was envisioned primarily as a means to validate the mesoscopic level. This was later found to be impractical and was not fully pursued.

Parameters used and the actual functional form of the path-integral Lagrangian are dependent on the actual system being modeled, that is, the brain and its neurons, or combat and its associated land, sea, or air battles. Ingber has already applied the concepts proposed here to both the combat scenario and the neocortex. The SMNC can be fit to many specific scenarios through manipulation of key variables discussed later in this chapter.

B. DOMAINS

Most research into neural nets fails to address the nonequilibrium situations that give rise to the thought process of the brain; the afferent convergence and efferent divergence of neural impulses. To properly account for this, it is necessary to introduce the notion of scaling, or domains. If brain organization is to be taken seriously, to adapt to other systems, it is reasonable to expect that some properties of the real brain be calculated by a theory.

1. The Microscopic Domain

Literature in the field of biological intelligence [5, 11, 21, 26, 27] suggests a statistical mechanical approach to modeling the macroscopic regions of the brain, specifically the neocortex, by statistically aggregating its microscopic regions or neurons. Many properties of biological and physiological nonlinear nonequilibrium systems can be modeled through the application of statistical mechanics. In the SMNC, the physiological properties of the neural system within the brain are approximated through the application of a nonlinear dynamic assemblage of quasi-random decision element, characterized by a complex pattern of variable neuron to neuron connections.

A typical neuron collects signals from its environment continuously summing them while deciding whether or not to respond based upon some threshold limit. When describing the large collection of neurons within the neocortex, it is postulated [5] that the brain averages the incoming inhibitory (I) and excitatory (E) polarizations occurring at the base of the axon prior to determinating whether to fire. The decision to fire is stochastically determined from the strength of stimuli reaching the

neural site during a specific refractory period τ of 5 - 10 msec. If the strength exceeds a neuron's threshold value the neuron may fire.

The influencing factors for a neuron's firing are simulated by the SMNC through the derivation of normal distribution functions. These functions take as inputs known ranges of values for parameters (V_j, v_{jk}, ϕ_{jk}, A_{jk}, B_{jk}, N, N^*). A detailed discussion of these parameters is given by Ingber in his papers on Statistical Mechanics of Neocortical Interactions [4, 5, 22-25] and are discussed in general terms below.

V_j is used to denote a neuron's threshold potential. This is separately determined for each micronode during program initialization routines by using a Gaussian distribution centered about known threshold values ($10\,mV$) for the neocortex [5]. After initialization, these values are allowed to slowly change dynamically ("plastically").

The variable v_{jk} represents the net electrical potential observed at receiving micronode j during an interaction with sending micronode k. ϕ_{jk} represents the variance of the net electric potential and is a Gaussian distributed variate selected from a limiting range 0.09 to 0.11. The variable v is also modeled as a Gaussian distributed variate with values selected from a range of $\pm$ 0.1mV. A positive value for v_{jk} is taken for an excitatory response from neuron k, and a negative value for an inhibitory response. Initial values are derived from existing literature on the brain [28].

A_{jk} is the activity that is induced at micronode j when micronode k fires and is a Gaussian distributed random number that is chosen between 0.001 and 0.01. A_{jk} multiplied by the number of individual action potentials, v_k to approximate the threshold value v_{jk}. B_{jk} is distributed and selected similarly to A_{jk} and represents the

background noise influencing micro to micro interactions. These values have been derived from research by Ingber on the neocortex [5].

σ_j refers to a neuron's most recent firing state. σ_j can take on a value of either +1 indicating the neuron has recently fired, or -1 to indicate that it has not. The probability p_{σ_j} that a neuron j fires is derived from an exponential function that combines and normalizes the sum of the aforementioned inputs to neuron j. The SMNC calculates p_{σ_j} through the generation of the above variables and then compares this value to a random number uniformly distributed between zero and one. If p_{σ_j} exceeds the random variate, the neuron is said to have fired, otherwise it does not fire and the process of stimulation begins again. As with the brain, there is no situation where firing can be assured.

The variables defined above are combined in Equation 3.1 which expresses the probability that a given micronode will fire. The result is then compared with a variate uniformly distributed between 0.0 and 1.0. If the value of p_{σ_j} exceeds the value of this second variate, then that micronode fires.

$$p_{\sigma_j} = \frac{\exp(-\sigma_j F_j)}{\exp(F_j) + \exp(-F_j)} \tag{3.1}$$

where

$$F_j = \frac{V_j - \sum_k (a_{jk} v_{jk})}{[\pi \sum_k a_{jk} (v_{jk}^2 + \phi_{jk}^2)]^{1/2}} \tag{3.2}$$

and

$$a_{jk} = \frac{1}{2}A_{jk}(\sigma_k + 1) + B_{jk} \tag{3.3}$$

Although studies reveal several kinds of inter-neuronal relations, we will be interested in only the E and I interactions which can be aggregated into a Gaussian distribution for the interactions across the synaptic regions, and applied to statistical mechanical modeling techniques simulating the activities of the neuron to neuron connection. The mathematics behind the equations just discussed are depicted graphically in Figure 3.1 [6].

2. The Mesoscopic Domain

As in most nonequilibrium systems, a mesoscopic scale is required to permit the formulation of a statistical model for a microscopic domain. This also allows

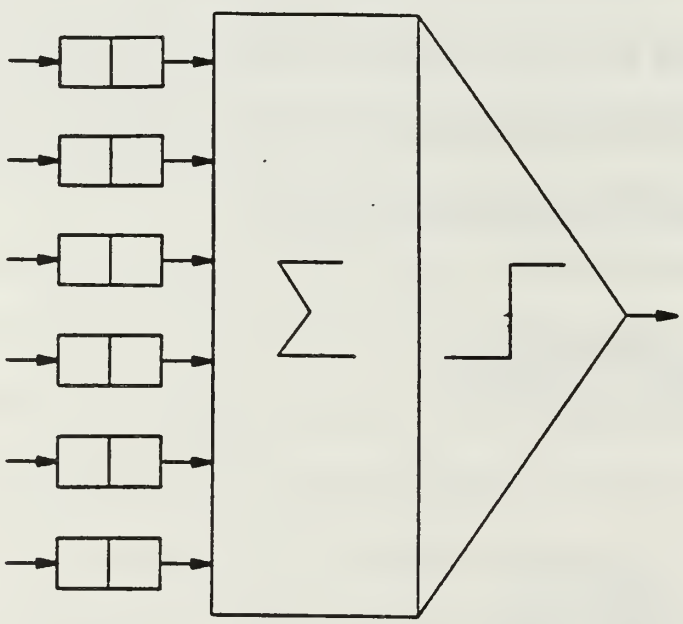

Figure 3.1 A typical neural unit.

the formulation of a macroscopic domain [29]. Within the brain, a physical scale exists supporting the concept of mesoscopic regions. Ingber [5] defines a vertically oriented collection of about 110 neurons as a minicolumn. These assemblages are found throughout the neocortex. Most neural interactions are short ranged, diverging through efferent minicolumns to as many as 10^4 nearest neighbors.

A mesocolumn can be thought of as an afferent minicolumn receiving inputs from approximately 10^4 other neurons. It is also viewed as an efferent macrocolumn scaled down to minicolumn size to relate the convergence/divergence of neocortical interactions. As with the brain, this permits a signal to be quickly propagated throughout a region [22]. Incoming signals are averaged and a single response is generated for transmission to as many as 10^4 other neurons.

Different cases of mesoscopic neural firings have been grouped together for separate consideration and modeling [5]. A model of dominant inhibition has been derived to explain the mechanisms behind the suppression of minicolumnar firings by neighboring minicolumns. Labeled IC, this model sets values for $A_{G'}^{G}$, the minicolumnar averaged conductivity between neuron G and neuron G', and $B_{G'}^{G}$, the minicolumnar averaged spontaneous background noise across the synaptic cleft between G and G'. The variable G is used to represent two possible classes of neurons of interest, E and I. A_I^E represents the case for the E-I, or excitatory to inhibitory connectivity. For the dominant inhibitory model this value is 0.01 N^*/N. The value N^*/N represents the minicolumnar weighting factor necessary to scale the averaging influence of 10^5 possible sources of stimuli in a macro region or macrocolumn, down to the minicolumn scale of 110. N, the number of neurons in a minicolumn, is the sum of

the N^E and N^I neurons which is about 110. N^*, the total number of neurons in a macrocolumn, is the sum of N^{*E} and N^{*I} neurons, which is equal to 10^3N, or approximately 10^5 neurons. It should be noted that there are four possible interactions (E-I, E-E, I,E, I-I). Research [25] also indicates that the average minicolumn has a predominance of the E type neurons over the I type neurons, where N^E is about 80 and N^I approximately 30. Values for all cases discussed are included in Table 3.1 which follows this chapter. Values from the table are for the microscopic case and need to be scaled by a factor N^*/N whenever applied to the mesoscopic scale.

In Ingber's works describing the results of stationary solutions of the macroscopic prepoint discretized Lagrangian L^G, it was noted that several minima clustered about the origin under sensitive changes of background noise, which he called a "centering mechanism [24]. Explanations of this clustering suggest that E-I competition at the mesoscopic scale produce a special case of the IC model, labeled IC' for dominant inhibition centered [5]. The values for A do not change in the IC' model. However, to account for the observed centering effect, Ingber [25] found it is necessary to change the values used for certain cases of B, specifically the balanced centered cases labeled B'^I_I where the value used is 0.0153 and B'^E_E which equals 0.00138.

At the other end of the scale of inter-neuronal responses lies the dominant excitation model labeled EC, which accounts for the nearest neighbor excitatory influences on a minicolumn. Applying the centering concept to the EC model give rise to what is labeled the EC' model or dominant excitation centered. This is accomplished by changing the parameters for B^I_E and B^E_I.

An intermediate set of models is used to account for the situation arising when the SMNC produces results in a range between excited and inhibited called balanced, or BC, and balanced centered, or BC'. These cases can be obtained by setting the listed values for A and B.

Through statistical manipulations of the microscopic region, mesoscopic parameters can be derived which reflect the net effect millions of neurons have with respect to their interconnections. This leads to the development of the probability distribution of mesocolumnar firings P. The value of P can be approximated from Equation 3.4

$$P \approx (2\pi\Delta t)^{-1/2} g^{1/2} \exp(-N\Delta t L) \tag{3.4}$$

where the variables are defined below.

Mesocolumnar interactions occur over the same interval τ as the neural interactions. N is the number of neurons in the mesocolumn, about 110. The mesoscopic Lagrangian L is computed as follows:

$$L^G = (2N)^{-1}(\dot{M}^G - g^G) g_{GG'} (\dot{M}^{G'} - g^{G'}) + M^G J_G/(2N\tau) - V'^G \tag{3.5}$$

Note the use of the Einstein convention of summing over repeated indices. g in Equation 3.4 is the determinant of the matrix $g_{GG'}$.

M^G represents the number of a particular type of neuron. Recall G may represent either an E or an I. M^E represents the number of excitatory neurons in a mesocolumn and can range from -80 to +80. M^I is the number of inhibitory neurons in the region and can take values between -30 and +30. $\dot{M}$ represents the time rate of change in M between sampling intervals or firings. Initial values for these parameters

are found by calling a random number generator. Later M is allowed to change dynamically within the program.

g^G represents the mean or first moment of the Lagrangian and $g^{GG'}$ is its variance or second moment. It is calculated in Equation 3.7. From this we are able to calculate values for four different cases: g^{EE}, g^{EI}, g^{II}, and g^{IE}. It should be noted that for the SMNC, $g_{EI} = g_{IE} = 0$ which reduces the calculation of g to a simple product shown in Equation 3.7 [5].

$$g = \det(g_{GG'}) \tag{3.6}$$

and

$$g^{GG'} = (g_{GG'})^{-1} = \tau^{-1} N^G \operatorname{sech}^2 F^G \tag{3.7}$$

Additionally:

$$g^G = -\tau^{-1}(M^G + N^G \tanh F^G) \tag{3.8}$$

The value of F^G is found using Equation 3.9.

$$F^G = \frac{(V^G - \sum_{G'} a_{G'}^G \nu_{G'}^G N^{G'} - \sum_{G'} \frac{1}{2} A_{G'}^G \nu_{G'}^G M^{G'})}{(\pi \sum_{G'} [(\nu_{G'}^G)^2 + (\phi_{G'}^G)^2](a_{G'}^G N^{G'} + \frac{1}{2} A_{G'}^G M^{G'}))^{1/2}} \tag{3.9}$$

where

$$a_{G'}^G = \frac{1}{2} A_{G'}^G + B_{G'}^G \tag{3.10}$$

The values for $a_{G'}^G$ may be computed using the data included in Table 3.1. The other unknown values in this equation are related to the mesoscopic domain and are as previously discussed; $\nu_{G'}^G = \nu^G$ which is approximately $0.1mV$, and $\phi_{G'}^G = \phi^G$ which is also about $0.1mV$. The calculations behind Table 3.1 can be demonstrated from

Equation 3.11 which solves for the synaptic background noise in the dominant inhibition centered case labeled IC'

$$B'^{G}_{E} = \frac{[V^{G} - (\frac{1}{2}A^{G}_{I} + B^{G}_{I})\nu^{G}_{I}N^{I} - \frac{1}{2}A^{G}_{E}\nu^{G}_{E}N^{E}]}{\nu^{G}_{E}N^{G}} \tag{3.11}$$

where both G is both E and I.

The Lagrange multipliers J_G represent inputs from interactions outside the macrocolumn. This simulates interactions between neurons across regions of the neocortex via long-ranged axons and is used as a means of influencing the mesocolumn through the direct input of external factors in much the same way a battlefield commander can be influenced by events outside his immediate scope. This factor is one deserving further investigation; however, it will not be evaluated in this thesis due to academic time constraints. The value of J_G used in all cases was zero.

V'^{G} is a mesocolumnar weighting factor and is computed using

$$V'^{G} = \sum_{G} V''^{G}_{G'}(\rho \nabla M^{G'})^2 \tag{3.12}$$

where ρ is the physical extent of the mesocolumn, or about 0.1 millimeter. $V''^{G}_{G'}$ values are also related to nearest neighbor interactions [4].

Test values of M^G (t + Δ t) are obtained from Cauchy distributions. Equations 3.5 - 3.12 are then employed to obtain a test value for P, the probability of firing. Next, the method of rejection is used to test for acceptance of the test value. A pseudo random number uniformly distributed between 0 and 1 is compared to the test value for P. Based of the this test P is either accepted for use by the SMNC, or it is

rejected, new test values of M^G are calculated by the Cauchy distribution, and a new value for P is calculated. This procedure is repeated until an acceptable probability of firing is found.

C. CONCLUSIONS

In the remainder of this thesis, we discuss the verification of the coding that was undertaken to ensure its accuracy as well as its ability to solve for the nonlinear probability distributions which describe the brain or other interesting systems. Ingber's papers on SMNI [4, 5, 22, 23, 25] have shown that mesocolumnar interactions can be accurately modeled using nearest-neighbor interactions. This permits accurate modeling of the neocortex while allowing a significant reduction in the number of calculations required for a single interaction. This feature has a significant influence over the usefulness of the SMNC as a tool for battlefield management, scientific research, or SDI.

TABLE 3.1

$A_C^{G'}$ $B_C^{G'}$ VALUES	
IC dominant inhibition	
$A_E^E = 0.005$ $A_E^I = 0.01$ $A_I^E = 0.01$ $A_I^I = 0.001$	$B_E^E = 0.001$ $B_E^I = 0.002$ $B_I^E = 0.002$ $B_I^I = 0.0002$
IC′ dominant inhibition centered	
$A'^E_E = 0.005$ $A'^I_E = 0.01$ $A'^E_I = 0.01$ $A'^I_I = 0.001$	$B'^E_E = 0.00138$ $B'^I_E = 0.002$ $B'^E_I = 0.002$ $B'^I_I = 0.00153$
EC dominant excitation	
$A_E^E = 0.01$ $A_E^I = 0.005$ $A_I^E = 0.005$ $A_I^I = 0.001$	$B_E^E = 0.001$ $B_E^I = 0.002$ $B_I^E = 0.002$ $B_I^I = 0.0002$
EC′ dominant excitation centered	
$A'^E_E = 0.01$ $A'^I_E = 0.005$ $A'^E_I = 0.005$ $A'^I_I = 0.001$	$B'^E_E = 0.001$ $B'^I_E = 0.002$ $B'^E_I = 0.0102$ $B'^I_I = 0.00862$
BC balanced	
$A_E^E = 0.005$ $A_E^I = 0.005$ $A_I^E = 0.005$ $A_I^I = 0.001$	$B_E^E = 0.001$ $B_E^I = 0.002$ $B_I^E = 0.002$ $B_I^I = 0.0002$
BC′ balanced centered	
$A'^E_E = 0.005$ $A'^I_E = 0.005$ $A'^E_I = 0.005$ $A'^I_I = 0.001$	$B'^E_E = 0.000438$ $B'^I_E = 0.002$ $B'^E_I = 0.0102$ $B'^I_I = 0.00862$

IV. OPERATION OF THE SMNC

The SMNC, as originally conceived, is composed of two computers operating in parallel, with the algorithms and theory of Chapter III pertaining to both. One computer models 10^5 neural units and operates at the mesoscopic, or middle scale. This scale takes advantage of the statistical mechanical shortcuts that are fundamental to this thesis and forms the basis of Ingber's derivation of short-term memory in the neocortex [5]. The second computer operates at the microscopic level and is a simulation of a fully connected neural computer made up of approximately 550 neural units. Each of these microscopic neural units is connected stochastically to about 10% of its neighbors. The microscopic computer was designed to serve as the basis for microscopically sampling the mesoscopic computer. Its purpose was to serve as a means to verify the mesoscopic output and provide a relative measure of speed and accuracy or resolution of the mesoscopic SMNC. Due to the size of memory required to just initialize the microscopic scale, the time required for even one update cycle, and the limited time available for thesis work this portion of the research was replaced by a more economical method of verification discussed later in this chapter.

A. INTRODUCTION

At the microscopic level the computer does not operate in complete isolation. A connectivity matrix is required to allow the individual neural units to communicate with the the other units at the same level of scale. This immediately presents a problem; interconnections with 10^5 neighbors must be simulated since a true neural computer of

this size would require over 5.5×10^6 interconnections. A connection matrix of this size would defeat the purpose of this thesis, quickly exhausting any reasonable computer resources. For this reason, the control portion of the project was approximated by using only five fully connected mesoscopic units consisting of 550 individual neural units. Even at this level of connectivity, the ability to run the microscopic neural computer at the Naval Postgraduate School is limited by the computing resources available (VAX 11/785). This necessitated the running of the microscopic scale of the SMNC on a more powerful computer. A VAX 8800 at NASA AMES, San Jose, CA was utilized for this purpose. Due to academic time constraints, the microscopic neural computer was run only to initialize the connection arrays and for one update cycle. This was done to ascertain the approximate memory requirements and running times for this portion of the project. Results of this part of the project are mentioned in Chapter V.

The SMNC models the neocortex region of the human brain. This modeling demonstrates the mesoscopic scaling algorithms, and highlights their utility in reducing the computational load associated with virtual neural computers. It also shows how the mesoscopic scale can be used to serve as an efficient filter between the microscopic and macroscopic scales, similar to the way information is filtered as it passes through the chain of command in a military organization.

1. The Microscopic Scale

The microscopic scale is the level at which an individual unit communicates with its neighbors. Traditional neural net computers only consider interactions at this level. However, in the SMNC, the determination of individual unit firing states at the microscopic level is done in parallel with operations at the middle or

mesoscopic scale. Calculations for the microscopic level may be carried out independently of those for the mesoscopic level. Because of the relatively small scale of the microscopic neural computer, its primary purpose was to sample the influence the context of the mesoscopic scale has on the microscopic.

2. The Mesoscopic Scale

Haken [26] points out the need for a mesoscopic scale in nonequilibrium systems to formulate the statistical mechanics of the microscopic system. This formulation permits development of the macroscopic scale and provide a means of filtering microscopic interactions. It also provides a channel for issuing "orders" in a C^3 application. The use of mesoscopic scaling also dramatically reduces the computational burden associated with neural computers.

B. COMPUTATIONAL EFFICIENCY

At the mesoscopic level, the SMNC makes use of several statistical techniques to reduce the "burden of computation". First, it is at this level that the computer employs the nearest neighbor concept to handle interactions that arise between mesoscopic groupings. The SMNC deals with macrocolumnar averaged minicolumns which can be viewed as having nearest neighbors. Figure 4.1 presents a generalized view of the nearest neighbor principle and how regions of influence overlap between units. In terms related to the neocortex, afferent minicolumns are represented by the small inner circles, outer circles sharing a common center with an inner circle represent macrocolumnar interactions developed by the minicolumn. The area outside the outer dark circle represents the number of efferent macrocolumnar nearest neighbor neurons. The inner circles represent the nearest neighbor interactions between the minicolumns. This

produces areas where information is shared between nearest neighbors and, through aggregation, the entire population.

The second statistical short cut is the scaling of the mesocolumns themselves. By aggregating the microscopic units in a mesocolumn and treating them as an afferent quantity, the SMNC is able to deal with groups of about 110 n_units as though they were single units. Instead of sampling each microscopic unit or neuron individually within a mesocolumn, the SMNC samples the mesocolumn, weighting its data accordingly.

These short cuts make the SMNC a practical tool for the battlefield commander to use in forecasting the course of events which will occur in his environment. They also provide motivation, at the research level, to run the two computers in parallel.

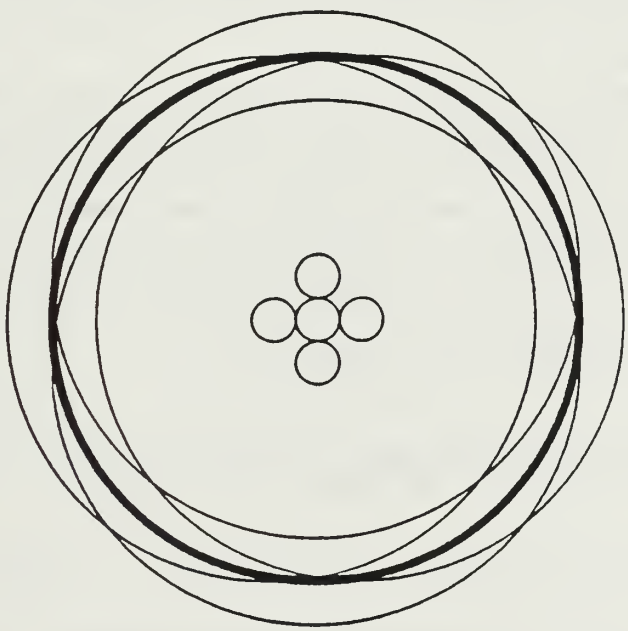

Figure 4.1 Nearest Neighbors.

At the mesoscopic level, the variable of interest is the M^G, which represents an output of the SMNC. Recall that M is the firing status of the mesocolumn, the value of which is tracked for both E and I type columns. The value of the Lagrangian L, which is a function of M^G, is also recorded for each unit change of time.

Initial values for the SMNC are of critical interest to this paper and the effects that slight changes in the values for M^G have on the running of the computer are discussed in Chapter V. For the first run, the values of the M^G are set so that 80% are at 0, 10% are at +1 and 10% at -1 as an arbitrary initialization of the system. The SMNC tracks through time with these variables assuming stable values while the trajectories reside in local minima. However, from time to time, the stochastic nature of the program will accept a variable far from an equilibrium point. Should it land near another of several existing local minima, the computer will track there until it is again forced to another metastable region.

In calculating the values of M^G, the SMNC makes use of several modified Monte Carlo techniques (discussed in Appendix C) to arrive at acceptable values. This is due to the interdependence of the variables and the sensitivity of the system to the initial conditions. Recall the equations for the Lagrangian, L.

$$L^G = (2N)^{-1}(\dot{M}^G - g^G)g_{GG'}(\dot{M}^{G'} - g^{G'}) + M^G J_G/(2N\tau) - V' , \tag{4.1}$$

where the value of $\dot{M}^G$ is derived as shown in Equation 4.2.

$$\dot{M}^G = [M^G(t+\Delta t) - M^G(t)]\Delta t . \tag{4.2}$$

The functional dependence of g^G and g_{GG} on M have been given previously (Equation 3.9). Given a value of M at time t, to arrive at $M(t+\Delta t)$ a test is conducted using the

Boltzmann method of rejection. A random number x, uniformly distributed on [0,1] is generated. DL, the difference in L values between update cycles, calculated as shown in Equation 4.3

$$DL = [(L_{trial}(s) + L_{trial}(s+1)) - (L_{old}(s) + L_{old}(s+1))]N\Delta t \tag{4.3}$$

where t = s(Δ t) + t_0. It can be seen that changes in M^G affect the values of two L's and thereby the values of $\sum DL$. If the value of the random variant, x is less than e^{-DL}, then the new M is accepted, otherwise no change in M is said to have occurred and its old value is retained.

Each update of the SMNC cycles through an entire micro-column, both spatially and temporally, calculating DL and updating the values of M^G for each spatial cell in each time increment Δt. These runs produce a trajectory through space and time for the values of M^G as time progresses. To test the results of the SMNC, several well known cases of non-linear systems whose solutions are known are examined. For this thesis, Ingber [5] provided a suitable application against which to test the results of the SMNC when applied to the neocortex case, introduced in Chapter III. Chapter V contains a discussion of the purpose of running the SMNC for the neocortical case. Work done with path-integral solutions to Fokker-Planck equations [30] provides another a means of validating the SMNC's ability to solve truly nonlinear probability distributions.

C. VERIFICATION

The purpose of verification is to ensure correctness of the algorithms and the concepts behind them. Verification of the SMNC required the existence of some test against which it can be run. Testing presents several complex problems since long term

probability distributions generally cannot be derived empirically without prior knowledge of the answer. Wehner and Wolfer [30, 31] faced similar difficulties in their work on path-integral solutions to Fokker-Planck equations.

They derived a numerical method, based on the path-integral formalism to solve nonlinear Fokker-Planck equations. In solving Fokker-Planck equations dealing with bifurcation of a stochastic process, Wehner and Wolfer used the drift function shown in Equation 4.4.

$$K(q) = \tanh(q) \tag{4.4}$$

The Lagrangian of this function is shown in Equation 4.5

$$\frac{(\dot{q}-\tanh(q))^2}{2} \tag{4.5}$$

with a constant diffusion coefficient equal to one.

Here, $\dot{q}$ is the difference between q(t) and $q(t+\Delta t)$. The long term solution to Equation 4.5 is known to be

$$P(q,t) = [\mathrm{sech}(q_0)/(2\pi t)^{1/2}]\exp(-t/2)\exp[(-1/2t)(q-q_0)^2]\cosh(q) \tag{4.6}$$

where $q_0 = 0$. A plot of this function at $t = 10.0$ is shown below in Figure 4.2 and can be seen to be the superposition of two Gaussian distributions with no nonzero steady state.

The purpose of the above was to introduce a Lagrangian which can be employed by the SMNC in an attempt to reproduce a known result, and to replace the time and memory intensive microscopic neural computer as the primary means of verification. The method of Monte Carlo integrations over the configuration space was proposed by Metropolis, *et al* [32] over 34 years ago as an approach to this problem. More recently,

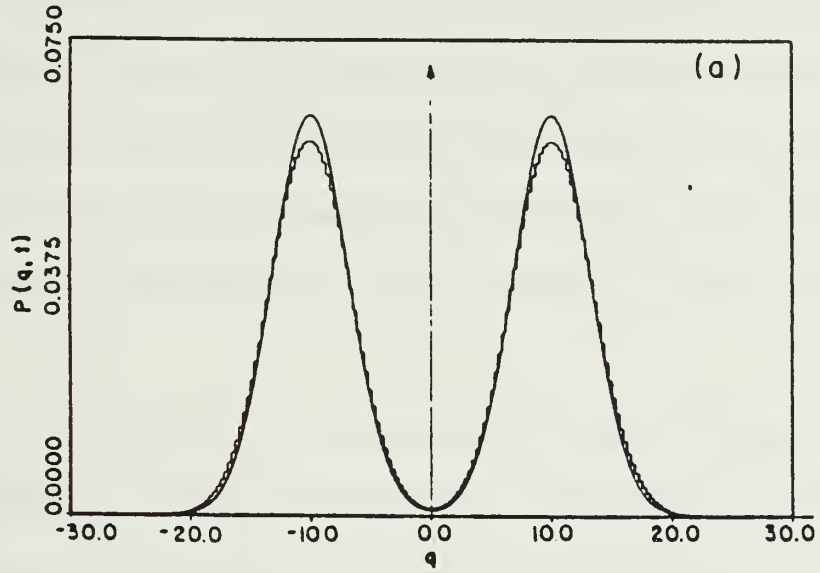

Figure 4.2 Graph of Equation 4.6 at time = 10

Landau [33] has applied Monte Carlo techniques to statistical mechanics. Landau states that to do a Monte Carlo simulation of a path-integral solution to a Fokker-Planck equation it is only necessary to be able to generate enough states according to the Boltzmann distribution to compute the properties which appear. The Boltzmann probability distribution can be written as

$$P = (2\pi\Delta t/g)^{-1/2} e^{-L\Delta t} = e^{-L\Delta t - 1/2\ln(2\pi\Delta t/g)} \tag{4.7}$$

where g is found from the following relationship.

$$g = \det(g^{GG}) \tag{4.8}$$

This is the approach taken by the SMNC to initialize its trajectories.

A modified Monte Carlo procedure which is driven by a Cauchy random number generator is employed where an "unlikely" first guess at the trajectory is made through

the use of a uniform random number generator. The values of this starting trajectory are then updated by generating random values with a Cauchy random number generator, and then using the Boltzmann distribution to test and accept or reject each new point as the trajectory changes. The Boltzmann test will ensure that the resulting distribution stays in the range of "likely" results. The use of a Cauchy random number generator allows for occasional testing of new space and thus for the possibilities of finding new local minima.

The entire trajectory is updated in this manner, say 1000 times, until the final product is free of transient variations. This resulting trajectory is now considered a "likely" distribution and is used as the actual initial starting condition for the SMNC. A more detailed description of this procedure is given in Appendix C.

The SMNC is run with Equation 3.5 replaced by Wehner and Wolfer's function. Using the Boltzmann test to arrive at a likely initialization point, Figure 4.3 was generated by the SMNC for the variable q over the same time interval as Figure 4.2. The results agree.

As a further check using the bifurcation test case, the initial condition, q_0, is set equal to 0.6. With elapsed time and time step the same as with the run which produced Figure 4.2, Wehner and Wolfer obtained a different set of results which are shown in Figure 4.4. The initial condition can be seen to have a great effect on the resulting probability distribution which is noted when the results obtained from the SMNC when applied to the same problem. Figure 4.5 was produced by the SMNC in solving the same problem. When the methods of computing the results are compared, it becomes significant that the SMNC is capable of such results in such a short amount of time.

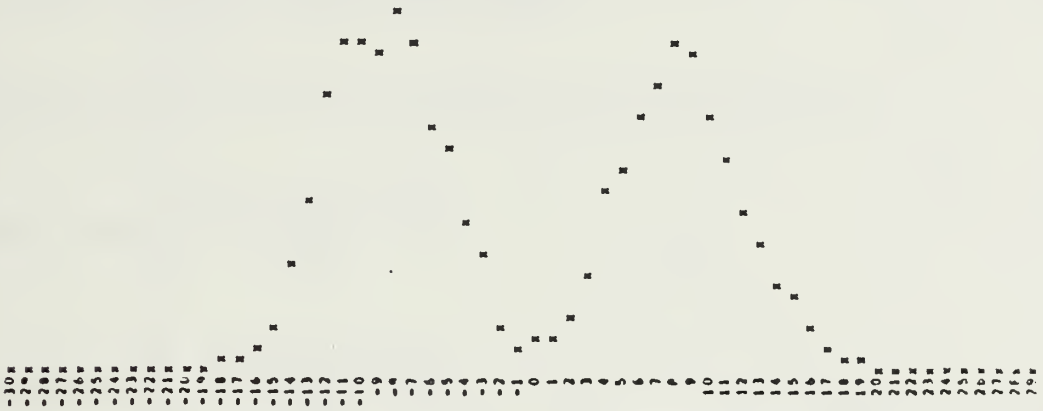

Figure 4.3 Plot of the SMNC's output for Equation 4.6.

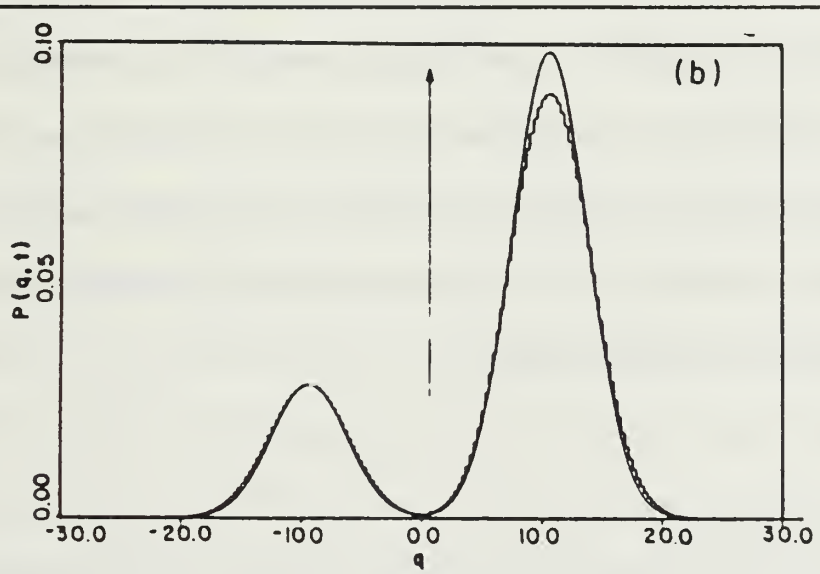

Figure 4.4 Wehner and Wolfer's solution to Equation 4.4 with $q_0 = 0.6$.

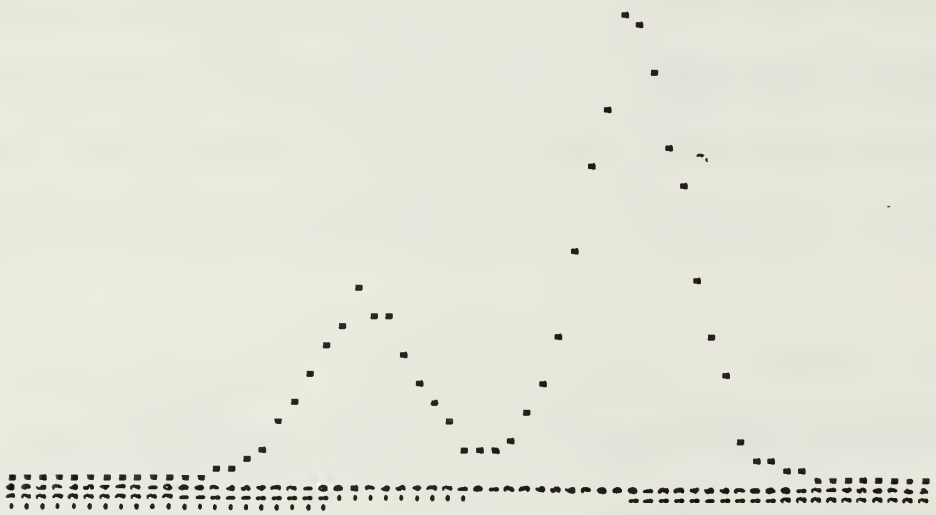

Figure 4.5 The SMNC solution to Equation 4.4 with $q_0 = 0.6$.

As a final test of the SMNC's ability to solve single-variable nonlinear probability distributions, an additional system, the Rayleigh gas model, was modeled for the verification process. The Rayleigh gas model consists of a dilute concentration of heavy atoms in a gas of lighter atoms. Treating these atoms as hard spheres, the Boltzmann equation for the ensuing collisions can be written as a Fokker-Planck equation. The drift function for this system is listed in Equation 4.9.

$$K = -q + 1.5 \tag{4.9}$$

The system diffusion function is:

$$Q = 2q \tag{4.10}$$

In this model, the Fokker-Planck equation is valid only for values of q greater than zero

since the energy of the gas particles can be only positive. The resulting energy distribution P(q,t) for the heavy particles, is given by Equation 4.11.

$$P(q,t) = \frac{e^{t/2}}{2[\pi q_0(1-e^{-t})]^{1/2}}\left[\exp[1]-\exp[2]\right] \tag{4.11}$$

The values of exp [1] and exp [2] found from Equations 4.12 and 4.13.

$$\exp[1] = \exp\left[-\frac{[q^{1/2}-(q_0e^{-t})^{1/2}]^2}{1-e^{-t}}\right] \tag{4.12}$$

$$\exp[2] = \exp\left[-\frac{[q^{1/2}+(q_0e^{-t})^{1/2}]^2}{1-e^{-t}}\right] \tag{4.13}$$

Figures 4.6 and 4.7 are for the short term probability distributions for the Rayleigh gas system. Wehner and Wolfer's solution to the Rayleigh gas system using the same initial condition ($q_0 = 7$) is shown in Figure 4.6. The SMNC was applied to the same problem with the results shown in Figure 4.7.

The long term probability distributions for the Rayleigh gas system are shown in Figures 4.8 and 4.9. Figure 4.8 is Wehner and Wolfer's solution for the system at time equal 10 minutes and same initial condition. The SMNC was applied to the Rayleigh gas system and produced the results seen in Figure 4.9 The similarities between these sets of figures and for those of the bifurcation cases in Figures 4.3 and 4.4 are taken as proof that the SMNC is capable of predicting the long term behavior of truly nonlinear probability distributions. Also of significance is the time and memory required to produce meaningful results. The methods of Wehner and Wolfer produce exact results and are fast. However, the computer resources required are far greater than those available to the battlefield commander. Their computations were made using a Cray super computer and

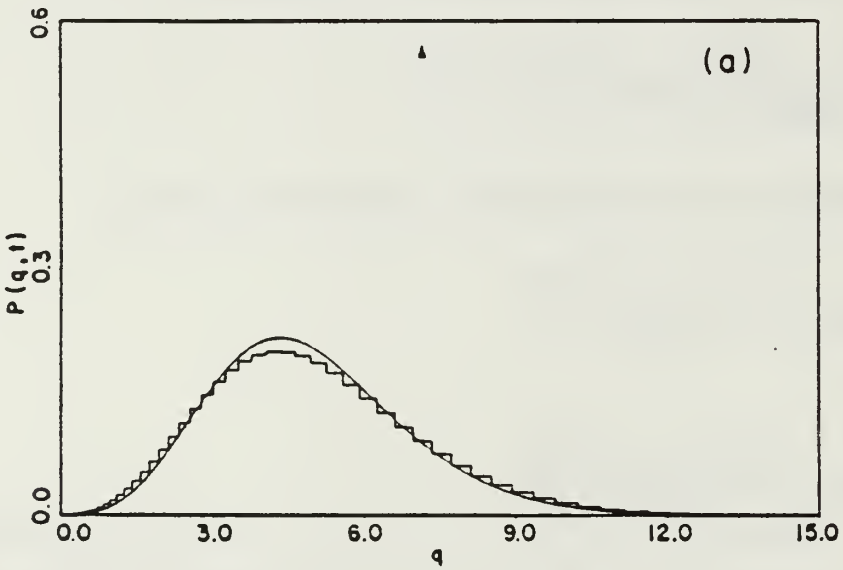

Figure 4.6 Rayleigh gas at time equals 0.5.

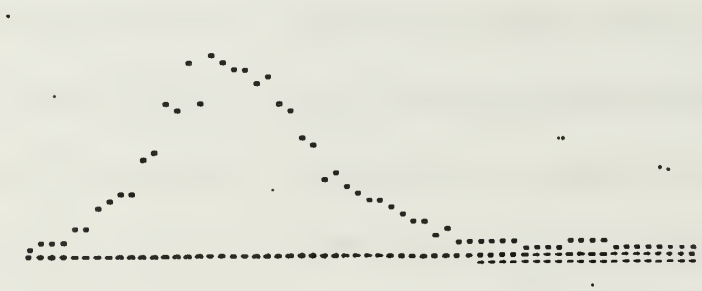

Figure 4.7 SMNC solution to Rayleigh gas system for time equals 0.5.

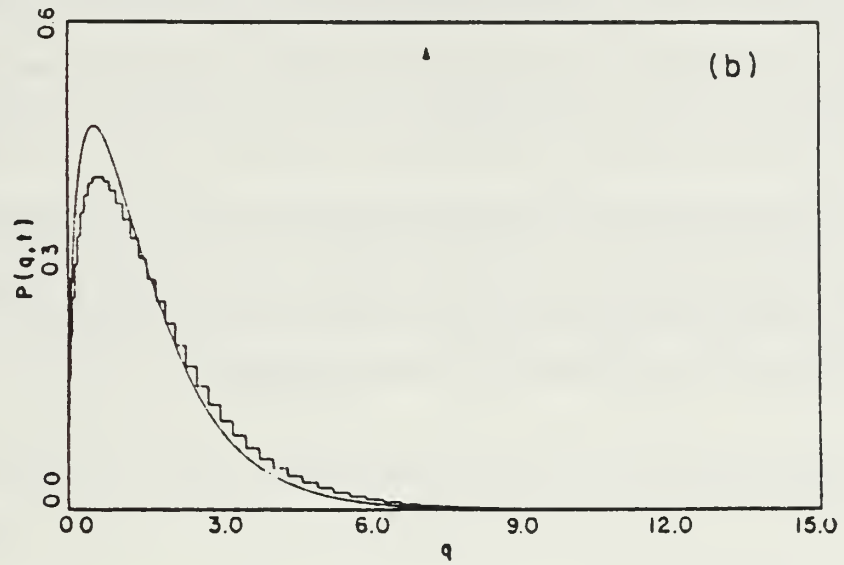

Figure 4.8 Wehner and Wolfer solution to Rayleigh gas at time equals 5.0.

Figure 4.9 SMNC solution to Rayleigh gas system for time equals 5.0.

required run times on the order of minutes. On the other hand, the SMNC derived a solution that was close in each case tested to those derived through more exhaustive methods of Wehner and Wolfer. Run times for the SMNC were longer, but also on the order of minutes. Any loss in resolution is a cost associated with the computational methods used. The source code used to produce the single variable case graphs shown of Figures 4.3 and 4.5, can be found in Appendix B. The user must decide whether or not a quick near fit is acceptable or the longer exact solution required.

V. CONCLUSIONS

Questions that may arise when discussing the SMNC should include the following:

- How much does the mesoscopic computer suffer from loss of resolution?
- Do the computational savings achieved by the mesoscopic computer compensate for any attendant loss of resolution?
- How closely does the mesoscopic computer correlate with the microscopic computer?
- How well does the mesoscopic computer filter data?
- How much can the SMNC learn?
- How robust is the SMNC?

Furthermore, now that the efficacy of the principles behind the SMNC has been demonstrated, additional research is required to build a real-time computer using state-of-the-art parallel processing techniques. Naturally, once a hardware implementation of the SMNC is available, other researchers may bring the SMNC capabilities to bear on C^3 problems in large-scale systems and data fusion, such as radar, sonar and electronic signals processing, missile guidance systems, and perhaps help in the development of an integrated battle management system.

A. INTRODUCTION

Work on the SMNC is just beginning, rather than coming to a conclusion. The groundwork laid by this thesis, and the many questions it and a related thesis by J.

Connell raise should provide the basis for further work in the area of neural computers. The increasing interest shown in the fields of neural computers has already lead to new break-throughs in architectural design of neural computer chips [34], the development of programming languages specifically designed for neural computers (Hecht-Neilsen Neurocomputer Corporation's *AXON*) and, the formation of the International Neural Network Society. In a paper presented at the National Defense University [35] considerable interest was generated in the SMNC and its ability to to deal with nonlinear probability distributions. This chapter seeks to answer some of its own questions, leaving several unanswered for others interested in the statistical mechanical approach to neural computers to solve.

B. SUMMATION OF RESULTS

The code required to run the mesoscopic SMNC (less than 300 lines in the C programming language) places it in the small program category. A skilled C programmer no doubt could further reduce this with a probable improvement in run time efficiency. However, the SMNC is an effective device for the preliminary study of nonlinear nonequilibrium probability distributions providing good results for the 1-dimensional cases and somewhat cruder results for 2-dimensional cases. Conventional procedures for conducting similar calculations typically employ programs of much greater sophistication and require much more CPU resources to produce results. As written, the code for the SMNC easily runs on a personal computer. With the proper Lagrangian, almost any scenario could be modeled.

An important factor in the calculations is the skill with which Lagrangians are derived and the accuracy of the data from which they are calculated. A limiting factor in

the resolution is the complexity of the Lagrangians and the amount of "crunching" the user has to devote to obtaining a solution. Therefore, when the question about computational savings is posed, the answer must consider the resources available, both in time and computing power. The battlefield commander may not be able to wait several hours for the best fit to his information. However, a solution which contains high quality data may be obtained in a matter of minutes with some acceptable loss of resolution.

The running of the microscopic computer for the purpose of comparing results with those obtained by the mesoscopic computer was not accomplished largely due to time constraints imposed by the magnitude of this procedure. The stated purpose of the microscopic computer was to validate the output of the mesoscopic computer. The Wehner and Wolfer solutions to the Fokker-Planck equations discussed in Chapter IV have enabled the validation of the code but not the use of the mesoscopic scale for the neocortical case. This, perhaps, would make an interesting thesis on its own. The neocortical case has also been programmed in the mesoscopic form by Professor Ingber in his efforts to confirm the validity of the SMNC. The results obtained thus far support the SMNC concept as a valid approach to the study of the neocortex and similar nonlinear systems.

The microscopic computer, described in Chapter III and Appendix A, was run once with the cooperation of the NASA AMES research facility in San Jose, CA. The results of this test run served only to confirm the magnitude of the computational burden required just to simulate five fully connected neural units. Running on a VAX 8800, a super computer capable of over 12 million floating point operations per second (flops), the microscopic neural computer of the SMNC required nearly 3 hours of devoted

computer time to initialize the connection arrays and execute one update cycle. For comparison, the mesoscopic computer run on the Naval Postgraduate School's VAX 11/785 (capable of 2 million flops) was able to initialize and update 100,000 times in the same length of time. The amount of memory required to store the connection array data was of similar proportions (over 2 giga bytes), further necessitating the assistance of a super computer. Of particular interest is the resolution that might be obtained from a program running at this scale.

Of equal interest is how closely does the microscopic scale model the human neocortex. Professor Paul Nunez, of the Biomedical Engineering Department at Tulane University, is currently pursuing this topic in collaboration with Professor Lester Ingber of the Naval Postgraduate School. Their goal is to better understand the nature of the neocortex and its roll in human recall.

When discussing how data is handled by the SMNC it should be noted that information is filtered in several ways. First, the use of a Cauchy distribution in conjunction with the Boltzmann test allows for wide variations in the accuracy of the data being analyzed by the SMNC. The Boltzmann test serves to dampen any oscillations away from the more likely trajectories while the Cauchy distribution permits sampling to continue at points that are not always within local minima. This effectively filters out data that does not fit the most likely trajectories while at the same time allowing the computer to look for other possible minima.

Losses in output resolution are related to two major sources. The coarse-resolution graphs from Chapter IV were developed using self-generated graphing

on a digital™ model LP14-DA line printer. The resolution was of sufficient quality to reproduce the characteristic bifurcation case, Figure 4.3, and gave very good results when applied to the Rayleigh gas system, Figures 4.5 and 4.7. For better resolution, graphics quality printers and routines must be employed. Computational resolution, in contrast to graphical resolution, is directly related to the number of update cycles, or time, available to the user for the development of stable trajectories. In the simple cases discussed in Chapter IV, running times were short enough to not be a factor. For more complex systems, such as the neocortex, the amount of time required for the system to stablize is much larger and results are more dependent on the number of trajectories tracked. When applying the SMNC to a system, the number of trajectories needed to satisfactorily produce results is one of the variables which will need to be determined prior to applying it to unknown tasks. For the single dimension case, between 10,000 and 50,000 trajectories were generated in reproducing the results of Wolfer and Wehner.

In answering the question, How much can the SMNC learn?, one must first show that the SMNC can learn. How much then becomes a matter for those interested in following the work completed so far. Learning is contained within the Lagrangians, fitted to specific systems by other procedures. Recall the equations from Chapter IV for the value of L:

$$L^G = (2N)^{-1}(\dot{M}^G - g^G)g_{GG'}(\dot{M}^{G'} - g^{G'}) + M^G J_G/(2N\tau) - V', \tag{5.1}$$

where the value of $\dot{M}^G$ is derived

$$\dot{M}^G = [M^G(t+\Delta t) - M^G(t)]\Delta t\ . \tag{5.2}$$

and for the value of DL:

---

digital is a trademark of Digital Equipment Corporation

$$DL = [(L_{trial}(s) + L_{trial}(s+1)) - (L_{old}(s) + L_{old}(s+1))] \quad (5.3)$$

In arriving at each value of DL, the SMNC must find an acceptable values for the next time step (via the Boltzmann-Cauchy procedure discussed in Chapter IV) using both previous (L_{old}) and projected (L_{trial}) values of L. The information contained in the old and new values of the Lagrangian provide the system with the means to use past data to influence future choices. The combination of past trajectories and the Boltzmann test for further trajectories ensures the system retains enough influence of the past to aid in decisions it makes about the future.

Robustness was not of primary concern during the development and testing of the SMNC. It is of concern to the battlefield commander applying it to a tactical decision in the field. Unfortunately no tests were conducted which addressed the robustness of the SMNC; however, as with the brain, the division of the decision-making elements into a vast number of interconnected and cooperating elements is estimated to result in a system that would be quite robust. This should be a topic of interest to those doing further research into neural-like processors.

C. VARIATION OF PARAMETERS

The effects several key parameters have on the outcome of the mesoscopic SMNC deserve mention. During the verification phase, several factors were adjusted with the affects on the firing distributions carefully noted. Those factors affecting the results were the temperature (variance used in the Cauchy routines), the number of trajectories plotted, the resolution, and the length of the warmup period used.

As mentioned before, the Boltzmann test uses a Cauchy distribution to sample points for possible new trajectories. The temperature sent to the Cauchy routine

determines the range of values the function returns. The larger the temperature, the wider the spread in values returned by the Cauchy routine. The Boltzmann test then checks the returned value for reasonableness of fit. The more a value deviates from the past set of trajectories, the more likely the Boltzmann test is to reject it. Therefore, a higher temperature will test points which fall away from the norm and result in a higher rejection rate. This is the best way to search for multiple minima when dealing with nonlinear probability distributions. The temperature should be large enough to sample the entire space a function is likely to be valid for.

A trajectory represents a possible path for a function over some discrete time interval. The function's value at the end of the time interval is the information we seek to learn about (for example the firing status of a neuron) and represents an output value for the SMNC. In using a Cauchy-driven Monte-Carlo method to generate each trajectory, the SMNC builds a set of trajectories which, in time, approach the long- term solution to the probability distribution it models. This is most easily seen from the figures in Chapter IV where the results presented represent an aggregation of between 50,000 and 100,000 trajectories. During the verification phase of this project, it was seen that the resolution was proportional to the number of terms (trajectories) aggregated, which in turn is determined by the limits of the output device.

In this context, the term resolution is used in conjunction with the display of data. The code written for the single-variable case included variables for the scaling of the output. The scale selected affected the results as would be expected. It was found that acceptable results could be obtained with a resolution chosen to use all the space on a 10

by 12 inch printout. This factor will not be discussed further since it is obviously dependent on the hardware being used.

The length of a warmup period is related to the number of trajectories used to generate results. During a warmup cycle, trajectories are generated with the resulting end points ignored until the completion of the warmup time. This method allows the system to generate enough trajectories to remove any initial bias caused by the randomly selected starting points and nonstable trajectories which follow. It was found, through the variation of this parameter, that 1000 was the minimum acceptable number of warmup cycles for the one-dimensional test problems.

D. CLOSING COMMENTS

The usefulness of the SMNC as a research tool is not yet fully understood. Its resolution is very coarse, although, in some cases sufficient to provide a researcher with an estimate of what some nonlinear function may look like during some point in time. This alone makes the SMNC a valuable research tool since in many cases in nature, such an estimate is beyond the reach of a simple program.

Appendix B contains the Cauchy-driven Monte-Carlo code for a one-variable Lagrangian. The coding and ideas presented in this paper, however, have already been applied to modeling combat Lagrangians as well as verifying the neocortex calculations. This required the development of a multi-variable Cauchy-driven Monte-Carlo code for nonlinear multivariate problems. Professor Ingber is currently applying these concepts to model both the neocortex and Janus simulated combat data. Initial results indicate the SMNC is capable of reproducing the results obtained from more exhaustive methods applied to the same systems.

As a final note, the trend in the military towards the dependence upon computer combat simulations has come under increasing criticism for several important reasons, among which are: the lack of real world data; the shrinkage of the time scales used in scenario evaluation; the increasing speed of evolution in real world tactics and logistics; and the increasing cost of computer simulations themselves. For all of these reasons, the requirement to improve the state of computer simulation becomes obvious. It can be argued that the code presented here, and developed further by Professor Ingber, provide a means of validating simulation data which is increasingly relied upon to help fill the demands of an ever changing world.

APPENDIX A: SOURCE CODE FOR THE NEOCORTICAL SMNC

Appendix A contains descriptions for some of the code for the microscopic SMNC (neocortical case) broken down into three major subsections: 1) the declaration of constants, variables, and the stochastic and memory allocation routines; 2) the initialization procedures for the data structures used; and 3) the update routines. Figures A.1 through A.14, at the end of this appendix, contain partial listings of the source code for the microscopic scale. As would be expected, the listing for this version of the SMNC is much longer than the version used to implement the mesoscopic system (for which source code is included with Appendix B). A discussion of each of the major code sections is contained within this appendix.

1. THE MISCELLANEOUS MODULE

The miscellaneous module of the SMNC contains declarations for all variables, constants, and data structures used to model the brain. Code for memory allocation is also included in this module. Initialization of variables for the SMNC is accomplished by the miscellaneous module. Variables controlling such things as length of the time slice, initialization of most recent firing status, numbers of excitatory and inhibitory n_units, etc., are all contained here.

In the discussion of statistical models, the user is required to first specify the class and characteristics of the underlying probability distributions being used. This must also include the space of the possible outcomes for the data being used. For this reason a brief discussion of the random number routines is also included in this section with code for several of the random number routines shown in Figure A.1.

a. Data Structures and the SMNC

The SMNC seeks to reproduce the workings of the human neocortex as closely as possible, hence the use of the term neural computer to describe the class of system created. To achieve this, several key data structures have been developed to perform the operations that transform randomness into meaning. A listing for the code for these structures may be found on Figure A.2. The data structures which model the neuron and associated clusters or mini columns are called n_units for the neuron case and micro_columns for the minicolumn case. Associated with both these entities is a data structure called a micro_connect, which holds the information about connectivities between n_units or micro_columns and strengths of such connections.

The n_unit has data fields which hold information about its class (recall that a neuron can be either excitatory or inhibitory). The n_unit also stores information about its firing threshold and most recent firing history. There is also a data structure within the n_unit which holds information about connections to other n_units called the micro_connect.

The micro_connect contains data which ultimately determines the firing rate for an n_unit. This includes the connectivity data discussed in Chapter 3; A_{jk}, B_{jk}, ν_{jk}, and ϕ_{jk}. There is also a pointer to the next micro_connect in a "chain" of connections that makes up the microscopic neural computer.

The micro_column models the mesoscopic scale of the brain. It contains several other data structures as well as fields for the many variables used in computing the Lagrangian L as shown in Equation 3.5. The micro_column also contains a "typical" neuron for both E and I classes. Recall one of the major reasons behind the SMNC's

mesoscopic scale is the savings in computation realized when dealing with a weighted "typical" n_unit as opposed to dealing with 1089 individual n_units.

b. Random Number Routines

In the generation of random numbers, the SMNC calls upon one of two pseudo random number generators to produce either a floating-point number or an integer. These routines modify two system-supplied macros, *rand* and *srand,* to create an array of random numbers that can be used and updated when needed.

The characteristic that distinguishes most distributions is the behavior of their tails. In this respect, the uniform distribution puts none of its outcomes outside two standard deviations. A Cauchy distribution puts between 25% and 30% of its outcomes beyond +/- two standard deviations and a Gaussian distribution will put about 4% of its outcomes beyond two standard deviations.

A uniform distribution places equal probability of an event occurring anywhere within a specified interval. On a given interval, [0,1] for example, divided into many equal parts, there is an equal probability, 1 divided by the number of subdivisions, that any particular partion will contain the event of interest. In the SMNC, uniform distributions are employed in the selection of most likely events where all the outcomes are equally likely to occur, such as during the initialization of a unit's firing status.

When the characteristic being modeled is actually a combination of linear events such that each event carries a small weight with respect to the total process, then a Gaussian distribution is the appropriate choice. The SMNC employs a Gaussian random number generator in the initialization routines for certain variables. The initial value found in the declaration module is sent to a Gaussian routine which returns a Gaussian

variant centered about this initial value. Inputs to a Gaussian distribution generator include estimates for the mean and variance for the property being modeled.

The Cauchy distribution is a symmetric stable distribution with a theoretically infinite tail; that is, it is possible to expect some "reasonable" number of values to occur at arbitrarily large distances from the mean. The SMNC uses a Cauchy distribution when an extreme spread of values are required. In the SMNC, a Cauchy distribution is used in conjunction with the method of rejection whereby values returned by a Cauchy distribution are then further tested against a uniform distribution. If the Cauchy value falls within the uniform value it will be accepted, otherwise the SMNC returns to the Cauchy routine for another candidate for use.

In addition to the declaration of all constants and variables, the following segments of code contain the random number and probability distribution functions which are used by the SMNC. The code is written in the C programming language and was implemented on a VAX 11/785 at the Naval Postgraduate School.

2. THE INITIALIZATION MODULE

The initialization module contains the code for the initialization of all of the data structures of the SMNC. This is shown in Figures A.3 through A.5. Each data structure exists to represent a physical section of the brain. This is segregation according to the level of scale being simulated. The data structure which models the minicolumn is the microcolumn and its associated initialization subroutine is init_microcolumn(). Init_typical() is used to establish a "representative" neuron of each class (E or I) inside each minicolumn. Finially, init_nunits() is the subroutine used by the SMNC to model the individual unit or neuron. All three of these modules use memory allocation routines

to set aside sufficient space in memory to hold one of these data structures. Since communication must take place between the various "units", these data structures are globally declared.

a. The Mesoscopic Scale

Initialization of the mesoscopic data structures consists of two main subroutines: init_typical() and init_microcolumn(). Most of the CPU time used by these modules is spent establishing the "identity" of the individual components and the connectivity that exists between them. Initial values for the variables used by the equations in Chapter 3 are determined through the use of a Gaussian generator which takes as inputs the initial values from the declarations module, and returns a Gaussian variant. Those few selected microcolumns which will serve as fully developed neural computers are designated separately and memory for their variables is set aside to be later filled in by init_nunit().

The microcolumns which are not chosen to be fully developed neural networks are given memory for the "typical" n_units which will be initialized by the subroutine init_typical() which is shown in Figure A.6. This procedure also uses a Gaussian distribution about preselected values to assign weighted values for the parameters of the mesoscopic structure.

b. The Microscopic Scale

In the SMNC five micro_columns are used as fully representative neural computers. Init_nunits(), shown in Figures A.7, A.8 and A.9, handles the initialization of the five special micro_columns. The "chosen five" are connected to all the typical n_units (both E and I), as well as to approximately 10% of their neighbors. With the

previous routines, a Gaussian procedure for selecting initial values of parameters is also employed. The subroutine which handles this scale is called init_microcolumn(). It is the magnitude of interconnectivity and the amount of data associated with each connection that greatly increases the running time and memory requirement of the SMNC. For this reason, the two computers do not usually run simultaneously on the VAX 11/785. To test the initialization of the of the microscopic level, the SMNC was run on a VAX 8800 with the results discussed briefly in Chapter V.

3. THE UPDATE MODULE

The SMNC can be viewed as two separate computers running simultaneously. The microscopic neural computer runs independent of the mesoscopic neural computer. Micro_update() handles the update cycle for the five fully-developed microcolumns. Likewise, meso_update() deals strictly with updating the meso_column. Each module therefore can be viewed as an independent program. After initialization has been completed, the program cycles through the data structures updating the firing status of the units, microscopic or mesoscopic.

a. The Microscopic Update Cycle

In the microscopic SMNC each n_unit "feels" the influence of every other n_unit within the system. This is accomplished through the use of several structures, each containing pointers to the other structures and exhaustive loops which test every n_unit. The requirement for each unit to know not only its own identity, or address, but also the address of all its neighbors leads to a major resource sink within this portion of the program. Further adding to the demands for memory and CPU time are the values for the variables needed to compute the firing status of each n_unit after each update cycle.

Assuming 1089 minicolumns, each having 110 n_units or neurons (equal to an area of approximately 1mm square in the neocortex) requiring some 20 variables (floats and integers), the growth of the required memory for initialization alone can be easily seen. This is also the major reason for the use of a "typical" neuron to represent an excitatory and inhibitory neuron in the minicolumns which are not truly represented. During an update cycle, each of the n_units samples every other n_unit during the process of determining whether or not to fire. This requires the calculation of the variables from Chapter III for each connection, which is a time consuming process. The microscopic update updates the "typical" n_units with weighted values for the variables from Chapter III. Figures A.10 through A.14 contain excerpts of code for the microscopic update routine, micro_update().

b. The Mesoscopic Update Cycle

In the mesocsopic update routine, each minicolumn has its excitatory and inhibitory n_units updated by the use of a Cauchy generator which employs the Boltzmann test for acceptance or rejection of possible updated values. Values for the variables of Equation 3.5 (or 4.2) are obtained in the same manner as with the microscopic scale. The result of the update cycle is a plot of the firing status of the mesoscopic scale over time. Figures A.13 and A.14 contain a listing for the subroutine get_L() which returns the value of the Lagrangian used in the mesoscopic update routine of the meso scale which coexists within the microscopic SMNC . This is included to permit the reader to make a comparison with the same routine for the mesoscopic SMNC described in Chapter IV and shown in Figure B.3.

```
float get_randf()
        {
        static int flag;
        static float f_bin[SHUFFLE];
        unsigned int i;
        int ran_dex;
        if (flag == 0)
                {
                srand(SEED);
                for (i = 0; i < 512; i ++)
                        rand();
                for (i = 0; i < SHUFFLE; i++)
                        f_bin[i] = (float) rand() / INTMAX;
                flag = 1;
                }
        ran_dex = rand() % SHUFFLE;
        f_bin[ran_dex] = (float) rand() / INTMAX;
        return(f_bin[ran_dex]);
        }

float gauss(mu, vare)
float mu, vare;
        {
        double x, sqrt();
        float get_randf();
        mu *= 100;
                {
                do
                        x = mu + (sqrt((-2 * sqrt(100 * vare) *
                                log(get_randf()))) * sin(2 * PI * get_randf());
                while(x <= 0.0);
                return ((float)x / 100);
                }
        }
float uniform(low_bound, hi_bound)
float low_bound, hi_bound;
        {
        return(low_bound + (hi_bound - low_bound) * get_randf());
        }
int cauchy(median, range)
int median, int range;
        {
        float get_randf() x;
        return((int)(range * tan(PI*(get_randf() - 0.5)) + median));
        }
```

Figure A.1

```
init_microcolumn()
        {
        unsigned int i, j;
        float temp_A, temp_B, gauss(), uniform();
        struct n_unit *unit_alloc();
        struct micro_column *column_alloc(), *mc;
        mes_pntr[0] = column_alloc();
        for(i = 0; i < MESOSIZE; i++)
                {
                mc = mes_pntr[i];
                if(i <= MESOROW -1)
                        {
                        mc->north = mes_pntr[MESOROW * (MESOROW -1) + i] =
                                column_alloc();
                        if(i == MESOROW - 1)
                                mc->east = mes_pntr[0];
                        else if(i == MESOROW -2)
                                mc->east = mes_pntr[0]->west;
                        else
                                mes_pntr[i + 1] = mc->east = column_alloc();
                        mes_pntr[i + MESOROW] = mc->south = column_alloc();
                        if(i != 0)
                                mc->west = mes_pntr[i-1];
                        else
                                mc->west = mes_pntr[MESOROW - 1] = column_alloc();
                        }
                else if(i < MESOSIZE - MESOROW)
                        {
                        mc->north = mes_pntr[i - MESOROW];
                        if(i % MESOROW == 0)
                                mc->west = mes_pntr[i-1]->south;
                        else
                                mc->west = mes_pntr[i - 1];
                        if(i % MESOROW == MESOROW - 1)
                                mc->east = mes_pntr[i - MESOROW +1];
                        else
                                mc->east = mes_pntr[i - MESOROW + 1] ->south;
                        if (i < MESOSIZE - 2 * MESOROW )
                                mes_pntr[i + MESOROW] = mc->south = column_alloc();
                        else
                                mc->south = mes_pntr[i-(MESOSIZE-2*MESOROW)]->north;
                        }
                else
```

Figure A.2

```
init_microcolumn()
        {
        unsigned int i, j;
        float temp_A, temp_B, gauss(),'uniform();
        struct n_unit *unit_alloc();
        struct micro_column *column_alloc(), *mc;
        mes_pntr[0] = column_alloc();
        for(i = 0; i < MESOSIZE; i++)
                {
                mc = mes_pntr[i];
                if(i <= MESOROW -1)
                        {
                        mc->north = mes_pntr[MESOROW * (MESOROW -1) + î] =
                                column_alloc();
                        if(i == MESOROW - 1)
                                mc->east = mes_pntr[0];
                        else if(i == MESOROW -2)
                                mc->east = mes_pntr[0]->west;
                        else
                                mes_pntr[i + 1] = mc->east = column_alloc();
                        mes_pntr[i + MESOROW] = mc->south = column_alloc();
                        if(i != 0)
                                mc->west = mes_pntr[i-1];
                        else
                                mc->west = mes_pntr[MESOROW - 1] = column_alloc();
                        }
                else if(i < MESOSIZE - MESOROW)
                        {
                        mc->north = mes_pntr[i - MESOROW];
                        if(i % MESOROW == 0)
                                mc->west = mes_pntr[i-1]->south;
                        else
                                mc->west = mes_pntr[i - 1];
                        if(i % MESOROW == MESOROW - 1)
                                mc->east = mes_pntr[i - MESOROW +1];
                        else
                                mc->east = mes_pntr[i - MESOROW + 1] ->south;
                        if (i < MESOSIZE - 2 * MESOROW )
                                mes_pntr[i + MESOROW] = mc->south = column_alloc();
                        else
                                mc->south = mes_pntr[i-(MESOSIZE-2*MESOROW)]->north;
                        }
                else
```

Figure A.3

```
        {
        mc->north = mes_pntr[i - MESOROW];
        if(i % MESOROW == 0)
                mc->west = mes_pntr[i - 1]->south;
        else
                mc->west = mes_pntr[i - 1];
        mc->south = mes_pntr[i - MESOROW * (MESOROW -1)];
        if(i < MESOSIZE - 1)
                mc->east = mes_pntr[i + 1];
        else
                mc->east = mes_pntr[i - MESOROW +1];
        }
mc->MsupE[0] = init_vars();
mc->MsupI[0] = init_vars();
mc->N_E = init_params(NESIZE);
mc->N_I = init_params(NISIZE);
mc->V_E = gauss(VMEAN, SVAR * VMEAN);
mc->V_I = gauss(VMEAN, SVAR * VMEAN);
mc->C_E = uniform(0.15, 0.25);
mc->C_I = uniform(0.15, 0.25);
mc->L[0] = 0;
mc->A_EE = gauss(10.0, SVAR * 10.0);
mc->A_II = gauss(0.1, SVAR * 0.1);
mc->A_EI = gauss(5.0, SVAR * 5.0);
mc->A_IE = gauss(5.0, SVAR * 5.0);
mc->phi_E = gauss(phiMEAN, SVAR * phiMEAN);
mc->phi_I = gauss(phiMEAN, SVAR * phiMEAN);
mc->v_E = gauss(vMEAN, SVAR * vMEAN);
mc->v_I = - gauss(vMEAN, SVAR * vMEAN);
mc->B_EE = ((mc->V_E - (0.5 * mc->A_EI + 2.0) *
        mc->v_I * mc->N_I) - ( 0.5 * mc->A_EE * mc->v_E
        * mc->N_E)) / (mc->v_E * mc->N_E);
if(mc->B_EE <= 0.0)
        {
        mc->B_EE = 1.0;
        mc->B_EI = ((mc->V_E - (0.5 * mc->A_EE + 1.0) *
                mc->v_E * mc->N_E) - ( 0.5 * mc->A_EI * mc->v_I
                * mc->N_I)) / (mc->v_I * mc->N_I);
        }
else
```

Figure A.4

```
                mc->B_EI = 2.0;
        mc->B_IE = ((mc->V_I - (0.5 * mc->A_II + 0.2) *
                mc->v_I * mc->N_I) - (0.5 * mc->A_IE * mc->v_E
                * mc->N_E)) / (mc->v_E * mc->N_E);
        if(mc->B_IE <= 0.0)
                {
                mc->B_IE = 2.0;
                mc->B_II = ((mc->V_I - (0.5 * mc->A_IE + 2.0) *
                        mc->v_E * mc->N_E) - (0.5 * mc->A_II * mc->v_I
                        * mc->N_I)) / (mc->v_I * mc->N_I);
                }
        else
                mc->B_II = 0.2;
        mc->a_EE = .5 * mc->A_EE + mc->B_EE ;
        mc->a_II = .5 * mc->A_II + mc->B_II ;
        mc->a_EI = .5 * mc->A_EI + mc->B_EI ;
        mc->a_IE = .5 * mc->A_IE + mc->B_IE ;
        mc->id_flag = -1;
        mc->mp = NIL;
        mc->Etypical.class = 1;
        mc->Etypical.status[0] = MESOSTART;
        mc->Etypical.status[1] = MESOSTART;
        mc->Etypical.thresh = gauss(VMEAN, VAR * VMEAN) / 1000;
        mc->Itypical.class = -1;
        mc->Itypical.status[0] = MESOSTART;
        mc->Itypical.status[1] = MESOSTART;
        mc->Itypical.thresh = gauss(VMEAN, VAR * VMEAN) / 1000;
        switch(i)
                {
                case 515:
                        mc->id_flag = 0;
                        matrix.numE[0] = mc->N_E;
                        matrix.numI[0] = mc->N_I;
                        matrix.totalN[0] = mc->N_E + mc->N_I;
                        matrix.root[0] = unit_alloc(matrix.totalN[0]);
                        matrix.parent[0] = 7;
                        break;
                }
        }
}
```

Figure A.5

```
init_typical()
{ unsigned int i, j;
 struct micro_connect *mee, *mei, *mii, *,mie, *connect_alloc();
 struct micro_column *mc;
 float gauss();
 for (i = 0; i < MESOSIZE; i ++)
        { mc = mes_pntr[i];
          mes_pntr[i]->Etypical.path = mee = connect_alloc();
          mes_pntr[i]->Itypical.path = mii = connect_alloc();
          for (j = 0; j < MESOSIZE; j ++)
                { if(i != j)
                        { mee->next = connect_alloc();  mii->next = connect_alloc();
                          mee->home = mii->home = j;
                          mee->homeid = -1;  mii->homeid = -2;
                          mee->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                          mii->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                          mee->v_jk = gauss(vMEAN, mee->phi_jk);
                          mii->v_jk = - 1 * gauss(vMEAN, mii->phi_jk);
                          mee->A_jk = gauss(mc->A_EE / 1000, VAR * mc->A_EE / 1000);
                          mee->B_jk = gauss(mc->B_EE / 1000, VAR * mc->B_EE / 1000);
                          mee->A_jk = gauss(mc->A_II / 1000, VAR * mc->A_II / 1000);
                          mee->B_jk = gauss(mc->B_II / 1000, VAR * mc->B_II / 1000);
                          mee = mee->next;  mii = mii-> next;
                        }
                }
                mei = mee;  mie = mii;
     for (j = 0; j < MESOSIZE; j ++)
                { if(i != j)
                        { mei->next = connect_alloc();  mie->next = connect_alloc();
                          mei->home = mie->home = j;
                          mei->homeid = -1;  mie->homeid = -2;
                          mei->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                          mie->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                          mei->v_jk = -1 * gauss(vMEAN, mei->phi_jk);
                          mie->v_jk = gauss(vMEAN, mie->phi_jk);
                          mei->A_jk = gauss(mc->A_EI / 1000, VAR * mc->A_EI / 1000);
                          mei->B_jk = gauss(mc->B_EI / 1000, VAR * mc->B_EI / 1000);
                          mie->A_jk = gauss(mc->A_IE / 1000, VAR * mc->A_IE / 1000);
                          mie->B_jk = gauss(mc->B_IE / 1000, VAR * mc->B_IE / 1000);
                          mei = mei->next;  mie = mie-> next;
                        }
                }
        }
}
```

Figure A.6

```
init_nunits()
{
unsigned int i, j, k, l;
struct micro_connect *mc, *connect_alloc();
struct n_unit *mp, *mj;
int *p, counter, count;
float gauss(), get_randf();
for(i = 0; i < MSIZE; i++)
        {  mp = matrix.root[i];
for (j = 0; j < matrix.totalN[i]; j ++)
                { if(get_randf() >= 0.35)
                        mp[j].class = 1;
                else
                        mp[j].class = -1;
                mp[j].status[0] = MICROSTART;  mp[j].status[1] = MICROSTART;
                mp[j].thresh = gauss(VMEAN, VAR * VMEAN) / 1000;
                mp[j].path = mc = connect_alloc();
                count = 0;
                for(k = 0; k < MESOSIZE; k++)
                        {
                        mc->next = connect_alloc();
                        mc->home = k;   mc->homeid = -1;
                        mc->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                        mc->v_jk = gauss(vMEAN, mc->phi_jk);
                        if(mp[j].class ==1)
                                {
                                mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_EE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->A_EE / 1000);
                                mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_EE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->B_EE / 1000);
                                }
                        else
                                {
                                mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_IE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->A_IE / 1000);
                                mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_IE /
                                        1000, VAR * mes_pntr[matrix.parent[i]]->B_IE / 1000);
                                }
                        count += 1;  mc = mc->next;
                        }
```

Figure A.7

```
        for(k = 0; k < MESOSIZE; k++)
                {
                mc->next = connect_alloc();
                mc->home = k;   mc->homeid = -2;
                mc->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                mc->v_jk = -1 * gauss(vMEAN, mc->phi_jk);
                if(mp[j].class == 1)
                        {
                        mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_EI / 1000,
                                VAR * mes_pntr[matrix.parent[i]]->A_EI / 1000);
                        mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_EI / 1000,
                                VAR * mes_pntr[matrix.parent[i]]->B_EI / 1000);
                        }
                else
                        {
                        mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_II / 1000,
                                VAR * mes_pntr[matrix.parent[i]]->A_II / 1000);
                        mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_II / 1000,
                                VAR * mes_pntr[matrix.parent[i]]->B_II / 1000);
                        }
                count += 1;   mc = mc->next;
                }
        for(l = 0; l < MSIZE; l++)
                {
                if(l != i)
                        {
                        counter = temp_array(matrix.totalN[i]);
                        p = mic_array;
                        for(k = 0; k < counter; k++)
                                {
                                mc->next = connect_alloc();
                                mc->home = *p++;
                                mc->homeid = l;   mc = mc->next;
                                }
                        mc->next = NIL;
                        }
                }
        }
for(j = 0; j < matrix.totalN[i]; j++)
        {
        mc = mp[j].path;
        for(k = 0; k < count ; k++)
                mc = mc->next;
        while(mc->next != NIL)
```

Figure A.8

```
                {
                mc->phi_jk = gauss(phiMEAN, VAR * phiMEAN);
                mc->v_jk = gauss(vMEAN, mc->phi_jk);
                mj = matrix.root[mc->homeid];
                if(mj[mc->home].class == -1)
                        mc->v_jk = -mc->v_jk;
                if(mp[j].class == 1)
                        if(mj[mc->home].class == 1)
                                {
                                mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_EE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->A_EE / 1000);
                                mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_EE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->B_EE / 1000);
                                }
                        else
                                {
                                mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_EI / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->A_EI / 1000);
                                mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_EI / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->B_EI / 1000);
                                }
                else
                        if(mj[mc->home].class == 1)
                                { mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_IE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->A_IE / 1000);
                        mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_IE / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->B_IE / 1000);
                                }
                        else
                                {
                                mc->A_jk = gauss(mes_pntr[matrix.parent[i]]->A_II / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->A_II / 1000);
                                mc->B_jk = gauss(mes_pntr[matrix.parent[i]]->B_II / 1000,
                                        VAR * mes_pntr[matrix.parent[i]]->B_II / 1000);
                                }
                mc = mc->next;
                }
        }
    }
}
```

Figure A.9

```
micro_update()
        {
        struct micro_connect *mc, *mp;
        struct n_unit mr, ml, *mm, *mj;
        unsigned int i, j, t;
        double exp(), sqrt();
        float A_jk, summ_two, summ_one, F_j, p_sigmaj, weight, get_randf();
        t = toggle;
        for(i = 0; i < MSIZE; i++)
                { mm = matrix.root[i];
                for( j = 0; j < matrix.totalN[i]; j++)
                        { mc = mm[j].path;
                        summ_one = summ_two = 0.0;
                        while(mc->next != NIL)
                                { if(mc->homeid < 0)
                                        { if(mc->homeid == -1)
                                                { weight =  mes_pntr[mc->home]->N_E / 10;
                                                if(weight <= 0.0)
                                                        weight = 1;
                                                mr = mes_pntr[mc->home]->Etypical;
                                                }
                                        else
                                                { weight = mes_pntr[mc->home]->N_I / 10;
                                                if(weight <= 0.0)
                                                        weight = 1;
                                                mr = mes_pntr[mc->home]->Itypical;
                                                }.
                                        }
                                else
                                        { mj = matrix.root[mc->homeid];
                                        weight = 1;
                                        mr = mj[mc->home];
                                        }
                                A_jk = weight * (.5 * mc->A_jk * (mr.status[t] + 1)
                                        + mc->B_jk);
                                summ_one += (A_jk * mc->v_jk);
                                summ_two += A_jk * ((mc->v_jk*mc->v_jk) +
                                        (mc->phi_jk * mc->phi_jk));
                                mc = mc->next;
                                }
```

Figure A.10

```
            F_j = (mm[j].thresh - summ_one) /
                    sqrt(PI * summ_two);
            p_sigmaj = (exp(-1 * mm[j].status[t] *
                    F_j)) / (exp(F_j) + exp(-1 * F_j));
            if(p_sigmaj > get_randf())
                    mm[j].status[1] = 1;
            else
                    mm[j].status[1] = -1;
            }
    for( j = 0; j < MESOSIZE; j++)
            { summ_one = summ_two = 0.0;
            mp = mes_pntr[j]->Etypical.path;
            while(mp->next != NIL)
                    { if(mc->homeid == -1)
                            { weight =  mes_pntr[mc->home]->N_E / 10;
                            if(weight <= 0.0)
                                    weight = 1;
                            ml = mes_pntr[mc->home]->Etypical;
                            }
                    else
                            {weight = mes_pntr[mc->home]->N_I / 10;
                            if(weight <= 0.0)
                                    weight = 1;
                            ml = mes_pntr[mc->home]->Itypical;
                            }
                    A_jk = weight * (.5 * mp->A_jk * (ml.status[t]
                            + 1) + mp->B_jk);
                    summ_one += (A_jk * mp->v_jk);
                    summ_two += A_jk * ((mp->v_jk * mp->v_jk) +
                            (mp->phi_jk * mp->phi_jk));
                    mp = mp->next;
                    }
            F_j = (mes_pntr[j]->Etypical.thresh - summ_one) /
                    sqrt(PI * summ_two);
            p_sigmaj = (exp(-1 * mes_pntr[j]->Etypical.status[t] *
                            F_j)) / (exp(F_j) + exp(-1 * F_j));
            if(p_sigmaj > get_randf())
                    mes_pntr[j]->Etypical.status[1] = 1;
            else
                    mes_pntr[j]->Etypical.status[1] = -1;
            mp = mes_pntr[j]->Itypical.path;
            while(mp->next != NIL)
```

Figure A.11

```
                    { if(mc->homeid == -1)
                            { weight =  mes_pntr[mc->home]->N_E / 10;
                            ml = mes_pntr[mc->home]->Etypical;
                            }
                    else
                            { weight = mes_pntr[mc->home]->N_I / 10;
                            ml = mes_pntr[mc->home]->Itypical;
                            }
                     A_jk = weight * (.5 * mp->A_jk * (ml.status[t]
                            + 1) + mp->B_jk);
                     summ_one += (A_jk * mp->v_jk);
                     summ_two += A_jk * ((mp->v_jk * mp->v_jk) +
                            (mp->phi_jk * mp->phi_jk));
                     mp = mp->next;
                     }
              F_j = (mes_pntr[j]->Itypical.thresh - summ_one) /
                     sqrt(PI * summ_two);
              p_sigmaj = (exp(-1 * mes_pntr[j]->Itypical.status[t] *
                            F_j)) / (exp(F_j) + exp(-1 * F_j));
              if(p_sigmaj > get_randf())
                     mes_pntr[j]->Itypical.status[1] = 1;
              else
                     mes_pntr[j]->Itypical.status[1] = -1;
              }
        }
}
```

Figure A.12

```
float get_L(num, ME, MI)
int num, ME, MI;
        {
        struct micro_column *mc;
        float F_E, F_I, g_E, g_I, g_EE, g_II, g, J_E, J_I, V_E, V_I, L, DL;
        int flagM, N, tog, M_dotE, M_dotI, M_E, M_I;
        unsigned int t;
        double cosh(), tanh(), sqrt();
        t = toggle;
        if (t == 0)
                tog = 1;
        else tog = 0;
        J_E = J_I = 0;
        mc = mes_pntr[num];
        F_E = (mc->V_E - (mc->a_EE * mc->v_E * mc->N_E +
                mc->a_EI * mc->v_I * mc->N_I) -
                0.5 * (mc->A_EE * mc->v_E * mc->MsupE[t] +
                mc->A_EI * mc->v_I * mc->MsupI[t]))
                / sqrt(PI * (((mc->v_E * mc->v_E) + (mc->phi_E * mc->phi_E)) *
                (mc->a_EE * mc->N_E + 0.5 * mc->A_EE * mc->MsupE[t]) +
                ((mc->v_I * mc->v_I) + (mc->phi_I * mc->phi_I)) *
                (mc->a_EI * mc->N_I + 0.5 * mc->A_EI * mc->MsupI[t])));
        F_I = (mc->V_I - (mc->a_IE * mc->v_E * mc->N_E +
                mc->a_II * mc->v_I * mc->N_I) -
                0.5 * (mc->A_IE * mc->v_E * mc->MsupE[t] +
                mc->A_II * mc->v_I * mc->MsupI[t]))
                / sqrt(PI * (((mc->v_E * mc->v_E) + (mc->phi_E * mc->phi_E)) *
                (mc->a_IE * mc->N_E + 0.5 * mc->A_IE * mc->MsupE[t]) +
                ((mc->v_I * mc->v_I) + (mc->phi_I * mc->phi_I)) *
                (mc->a_II * mc->N_I + 0.5 * mc->A_II * mc->MsupI[t])));
        g_E = (mc->MsupE[t] + mc->N_E * tanh(F_E)) / -TAU;
        g_I = (mc->MsupI[t] + mc->N_I * tanh(F_I)) / -TAU;
        g_EE = 1 / TAU * mc->N_E * 1 / (cosh(F_E) * cosh(F_E));
        g_II = 1 / TAU * mc->N_I * 1 / (cosh(F_I) * cosh(F_I));
        g = (1 / g_EE) * (1 / g_II);
        M_dotE = (ME - mc->MsupE[t]) / DELTA_T;
        M_dotI = (MI - mc->MsupI[t]) / DELTA_T;
        V_E = 0;
```

Figure A.13

```
mc->C_E * (((mc->MsupE[t] -
        mc->east->MsupE[t]) *
        (mc->MsupE[t] - mc->east->MsupE[t])) +
        ((mc->MsupE[t] - mc->west->MsupE[t]) *
        (mc->MsupE[t] - mc->west->MsupE[t])) +
        ((mc->MsupE[t] - mc->north->MsupE[t]) *
        (mc->MsupE[t] - mc->north->MsupE[t])) +
        ((mc->MsupE[t] - mc->south->MsupE[t]) *
        (mc->MsupE[t] - mc->south->MsupE[t])));
V_I = 0;
mc->C_I * (((mc->MsupI[t] -
        mc->east->MsupI[t]) *
        (mc->MsupI[t] - mc->east->MsupI[t])) +
        ((mc->MsupI[t] - mc->west->MsupI[t]) *
        (mc->MsupI[t] - mc->west->MsupI[t])) +
        ((mc->MsupI[t] - mc->north->MsupI[t]) *
        (mc->MsupI[t] - mc->north->MsupI[t])) +
        ((mc->MsupI[t] - mc->south->MsupI[t]) *
        (mc->MsupI[t] - mc->south->MsupI[t])));
        if(num == 0)
                {
                M_dotE = 0;
                M_dotI = 0;
                V_E = 0;
                V_I = 0;
                }
N = mc->N_E + mc->N_I;
L = ((M_dotE - g_E) * (M_dotE - g_E) / (2 * N * g_EE)
        + (mc->MsupE[t] * J_E / (2 * N * TAU)) - V_E)
        + ((M_dotI - g_I) * (M_dotI - g_I) / (2 * N * g_II)
        + (mc->MsupI[t] * J_I / (2 * N * TAU)) - V_I);
return;
}
```

Figure A.14

APPENDIX B: SOURCE CODE FOR THE MESOSCOPIC SMNC

Having verified the concept of the mesoscopic scale in Chapter IV, the extra code and computing resources required by the microscopic neural become an unnecessary burden and therefore have been replaced. This results in a powerful program of less than 400 lines of code. Appendix B contains the code segments for the SMNC as written for the mesoscopic single variable case used to reproduce the bifurcation graphics in Chapter IV. Because of the brevity of the code, it is not broken down into modules as was done with the neocortical SMNC discussed in Appendix A.

The mesoscopic SMNC takes full advantage of the scaling algorithms discussed in Chapters IV and V. This code, in two versions, was applied to nonlinear probability distributions. Employing a single variable, the mesoscopic neural computer was used as a means of verification. A two variable version was also employed to test the concept of the mesoscopic scale before applying it to the brain. This test used the Wehner and Wolfer bifurcation problem and applied it to two dimensions. That is, the same variables were used for each dimension in X-Y pairs. The results for this test have not been included due to the poor resolution obtained for initial runs and the lack of sufficient time to perfect the code. The multi-dimensional Lagrangian case has, however, been perfected by Professor Ingber, working in collaboration with this author and fellow researcher CDR J. Connell, at the Naval Postgraduate School. Figures B.1 through B.6 contain a complete listing of the bifurcation verification code using the mesoscopic SMNC.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#define INTMAX 2147483647
#define PI 3.14159265359
#define DELTA_T 0.5
#define NUM 20
#define NUM1 NUM + 1
#define WARMUP 1000
#define N 50000
#define V 10.0
#define H 6.0
#define X_INCHES 1.66666667
#define Y_INCHES 533.33333333
#define RESOLUTION 1.0
#define SCALE 60.0
#define TEMP 1.0
#define CAUCHY 30.0
#define INIT 0.0
#define SEED 6969.0
float seed, temp;
double log(), tanh(), sqrt(), sin(), tan(), cos(), cosh(), exp();
struct master
        {
        float q, K, Q;
        };
struct master trajectory[NUM1];
```

Figure B.1

This figure contains the declarations and constants for the mesoscopic SMNC. It also makes the declaration for a structure to contain the drift and diffusion variables used in calculating the Lagrangian values.

```
float get_randf()
        {
        double x;
        x = 16870.0 * seed;
        seed = x - 2147483647.0 * floor (x / 2147483647.0);
        return (seed / 2147483648.0);
        }

float cauchy(median, temp)
float median, temp;
        {
        float dum1, dum2, ratio;
        ratio = 2.0;
        while (ratio > 1.0)
                {
                dum1 = 2.0 * (get_randf() - 0.5);
                dum2 = get_randf();
                ratio = dum1 * dum1 + dum2 * dum2;
                }
        return(temp * (dum1 / dum2) + median);
        }

float get_K(q)
float q;
        {
        return(tanh(q));
        }

float get_Q(q)
float q;
        {
        return(1.0);
        }
```

Figure B.2

This page contains several procedures used during the initialization and update phases. The routine get_randf() returns a pseudo random floating point number. Cauchy is used to return a Cauchy distributed variant centered about "median" with a variance of "temp". The two "get" routines generate the drift and diffusion for the bifurcation case.

```
init()
        {
        unsigned int i;
        trajectory[0].q = INIT;
        trajectory[0].K = get_K(trajectory[0].q);
        trajectory[0].Q = get_Q(trajectory[0].q);
        for(i = 1; i < NUM1; i ++)
                {
                trajectory[i].q = INIT;
                trajectory[i].K = get_K(trajectory[i].q);
                trajectory[i].Q = get_Q(trajectory[i].q);
                }
        for(i = 0; i < WARMUP; i++)
                update();
        }

float get_L(q1, q2, K, Q)
float q1, q2, K, Q;
        {
        return((q1 - q2 - (K * DELTA_T)) * (q1 - q2 - (K * DELTA_T)) /
                (2.0 * Q * DELTA_T));
        }
```

Figure B.3

This figure contains the initialization routine, init(), and a function for calculating the Lagrangian, get_L. Note that for the single variable case, the starting points are initialized to zero.

```
update()
        {
        unsigned int i; int flag_q;
        float DL, L1, L2, L3, L4, q_prime, test_K, test_Q, x, y, cauch;
        cauch = CAUCHY;
        for (i = 1; i < NUM; i++)
                {
                L1 = get_L(trajectory[i].q, trajectory[i-1].q,
                        trajectory[i-1].K, trajectory[i-1].Q);
                L2 = get_L(trajectory[i+1].q, trajectory[i].q,
                        trajectory[i].K, trajectory[i].Q);
                q_prime = cauchy(trajectory[i].q, temp);
                while((q_prime <= -cauch) || (q_prime > cauch))
                        q_prime = cauchy(trajectory[i].q, temp);
                test_K = get_K(q_prime);
                test_Q = get_Q(q_prime);
                L3 = get_L(q_prime, trajectory[i-1].q,
                        trajectory[i-1].K, trajectory[i-1].Q);
                L4 = get_L(trajectory[i+1].q, q_prime,
                        test_K, test_Q);
                DL = L4 + L3 - L1 - L2 + (log(test_Q / trajectory[i].Q) / 2);
                flag_q = 0;
                if(DL < -25.0)
                        {
                        x = 1.0;
                        y = 0.0;
                        }
                else if(DL > 25.0)
                        {
                        x = 0.0;
                        y = 1.0;
                        }
                else
                        {
                        x = exp(-DL);
                        y = get_randf();
                        }
                if(x > y)
                        flag_q = 1;
                if(flag_q == 1)
                        {
                        trajectory[i].q = q_prime;
                        trajectory[i].K = test_K;
                        trajectory[i].Q = test_Q;
                        }
                L2 = get_L(trajectory[i+1].q, trajectory[i].q, trajectory[i].K, trajectory[i].Q);
```

Figure B.4

```
        }
        i = NUM;
        L1 = get_L(trajectory[i].q, trajectory[i-1].q,
                trajectory[i-1].K, trajectory[i-1].Q);
        q_prime = cauchy(trajectory[i].q, temp);
        while((q_prime <= -cauch) || (q_prime > cauch))
                q_prime = cauchy(trajectory[i].q, temp);
        test_K = get_K(q_prime);
        test_Q = get_Q(q_prime);
        L3 = get_L(q_prime, trajectory[i-1].q,
                trajectory[i-1].K, trajectory[i-1].Q);
        DL = L3 - L1;
        flag_q = 0;
        if(DL < -25.0)
                {
                x = 1.0;
                y = 0.0;
                }
        else if(DL > 25.0)
                {
                x = 0.0;
                y = 1.0;
                }
        else
                {
                x = exp(-DL);
                y = get_randf();
                }
        if(x > y)
                flag_q = 1;
        if(flag_q == 1)
                {
                trajectory[i].q = q_prime;
                trajectory[i].K = test_K;
                trajectory[i].Q = test_Q;
                }
}
```

Figure B.5

The previous two figures contain the listing for the update procedure. The code on Figure B.4 updates trajectories for all but the final case. The code on Figure B.5 updates only the final trajectory.

```
main ()
        {
        unsigned int i, j;
        int hist[60];
        float n_bin[60], q_scale, ratio, factor,  height;
        seed = SEED;
        temp = TEMP;
        ratio = Y_INCHES / X_INCHES;
        q_scale = (1.0 / RESOLUTION)  * (1.0 / H);
        factor = q_scale * V * ratio;
        init();
        for (i = 0; i < N; i++)
                {
                if(i % 1000 == 0)
                        init();
                update();
                hist[30 + (int)(trajectory[NUM].q)] += 1;
                }
        printf(");
        printf("   ");
        for(i = 0; i < (0.0375 * factor); i++)
                printf(" ");
        printf("|");
        for(i = 0; i < (0.0375 * factor) - 1; i++)
                printf(" ");
        printf("|");
        printf("0);
        for(i = 0; i < (SCALE / RESOLUTION); i ++)
                {
                printf("63d",   i - 30);
                n_bin[i] = (float)(hist[i] / (float)N);
                height = n_bin[i] * factor  / RESOLUTION;
                for(j = 0; j < height; j++)
                        printf(" ");
                printf("x");
                }
        }
```

Figure B.6

Figure B.6 contains code for the main program and includes a graphing routine for the results which are displayed in Chapter VI.

APPENDIX C: MONTE CARLO CALCULATIONS

One's first encounter with path integrals may also be his last. With the advent of the computer, however, the approach often taken to path integrals and similar non-linear problems is through the use of Monte Carlo methods. Although the spinning of a roulette wheel has been replaced by the random number generator, the simulation of a random process is still referred to as a Monte Carlo calculation.

An example used by Landau [33] in equilibrium statistical mechanics may demonstrate this point. Landau used the case of a heat reservoir in which the probability $P(\alpha)$ that the system is in the state α can be expressed by the following:

$$P(\alpha) = \frac{e^{\frac{-E(\alpha)}{kT}}}{\sum_{\alpha} e^{\frac{-E(\alpha)}{kT}}} \tag{C.1}$$

where k is Boltzmann's constant, T is the temperature of the reservoir, and $E(\alpha)$ is the energy for some state α. It should be noted that Equation B.1 applies no matter what the system state was before it made contact with the heat reservoir. Equation B.1 shows the random nature of interactions of the environment with the heat reservoir. Just as there may be many kinds of reservoirs in nature, there also may be many types of computer programs to simulate them.

Cauchy-driven Monte Carlo techniques are used by the SMNC to allow for stochastic changes within the system. These occur during the initialization of the system variables as well as during the calculations of the trajectories of the M^G variables through time. In an attempt to alleviate some of the problems the standard Monte Carlo techniques have with multi-minima Lagrangians (finding only local minima), the SMNC

also makes use of a Cauchy distribution to generate test points which will occasionally lie outside a relative minima. This allows the SMNC to sample more points within its environment, thereby giving it a greater possibility of finding multiple minima.

In regenerating the path-integral solutions to the Fokker-Planck equations done by Wehner and Wolfer [30], the following Lagrangian was calculated in the prepoint discretization.

$$L(t) = \frac{[(q(t+dt)-g(t))/dt - K(t)]^2}{2Q(t)} \tag{C.2}$$

The associated probability distribution is found from Equation B.3.

$$p(t) = [2\pi Q(t-dt)dt]^{-1/2} e^{-Ldt} \tag{C.3}$$

For this problem, time is discretized, but not necessarily q, the discretization of which depends on the system modeled. For the Wehner and Wolfer problem, Q(t) = 1 for all time and q(0) = 0. A Cauchy random number generator is used to supply test values of q(t) for times greater than 0.

The Boltzmann test uses a comparison between a uniform random number between 0 and 1, and B, which is calculated by Equation B.4.

$$B = e^{-DLdt} \tag{C.4}$$

where

$$DL = \sum_{L}(L_{new} - L_{old}) \tag{C.5}$$

and where the sum is taken over all L's affected by changes in the points generated by the Cauchy distribution. Here, new and old refer to entire trajectories.

L(t) includes the "temporal" nearest-neighbor interactions (which are also necessary for the neocortical problem) such that

$$\dot{q}(sdt) = \frac{[q((s+1)dt) - q(s)dt]}{dt} \tag{C.6}$$

where s ranges from 0 to n, and the the final time t is equal to (n + 1) times dt (both s and n are integers). This implies that only the values of L at neighboring times are needed to calculate DL. The SMNC accomplishes this through the generation of new trajectories by changing one q value at a time. The likely trajectory this produces is then accepted or rejected according to the Boltzmann test.

This procedure is repeated as often as necessary to produce a sufficient number of trajectories for the system to react to the Lagrangian. It was found in the case of the Wehner and Wolfer data that after as few as 100 trajectories, the system was able to "feel" all its temporal points. In the multi-variable, multi-spatial-cell cases (such as the brain or combat terrain) more trajectories are needed to allow all of the variables at all spatial-temporal points to interact. Even for the many variable case, however, the number of changes needed to be made to the code is small when compared to other statistical mechanical methods.

LIST OF REFERENCES

[1] Stevens, L., *Artificial Intelligence, The Search for the Perfect Machine* (Hayden, Hasbrouck Heights, NJ, 1985).

[2] "Introduction to Data Level Parallelism," Thinking Machine Technical Report 86.14, Thinking Machine Corp., Cambridge, MA, 1986.

[3] Hecht-Nielsen, R., *Performance Limits of Optical, Electro-Optical and Electronic Neurocomputers* (Hecht-Nielsen Neurocomputer Corp., San Diego, CA, 1987).

[4] Ingber, L., "Statistical mechanics of neocortical interactions. I. Basic formulation," *Physica D 5*, 83-107 (1982).

[5] Ingber, L., "Statistical mechanics of neocortical interactions. Derivation of short-term-memory capacity," *Phys. Rev. A 29*, 3346-3358 (1984).

[6] Brown, R. J., "An Artificial Neural Network Experiment," *Dr. Dobb's J. of Software Tools* 16-27 (April 1987).

[7] Stevens, C. F., "The Neuron," in *The Brain 'A Scientific American Book'*, ed. by D. Flanagan et al (W. H. Freeman, New York, 1979).

[8] Hubel, D. H., "The Brain," in *The Brain 'A Scientific American Book'*, ed. by D. Flanagan et al (W. H. Freeman, New York, 1979).

[9] Sampson, J. R., *Biological Information Processing* (John Wiley & Sons, New York, 1984).

[10] Clark, J. W., Rafelski, J., and Winston, J. V., "Brain without Mind: Computer Simulation of Neural Newtowks with Modifiable Neural Interactions," *Phy. Rep. 123*, 215-273 (1985).

[11] Bruckstein, A. and Yehoshua, Z., "An Adaptive Stochastic Model for Neural Coding Process," *IEEE Transactions on Systems, Man and Cybernetics SMC 15*, (1985).

[12] Rosenblatt, F., *Principles of Neurodynamics* (Spartan, Washington, DC, 1961).

[13] Minsky, M. and Papert, S., *Perceptrons* (MIT Press, Cambridge, MA, 1965).

[14] Green, L., ''Neural-Net Systems: Computers that Learn,'' *Information Week* 32 (16 March 1987).

[15] Abu-Mustafa, Y. F. and Psaltis, D., ''Optical Neural Computers,'' *Scientific American* 88-96 (March 1987).

[16] Williams, T., ''Optics and Neural Nets: Trying to Model the Human Brain,'' *Computer Design* 47-62 (1 March 1987).

[17] Hillis, W. D., *The Connection Machine* (MIT Press, Cambridge , 1985).

[18] Hopfield, J. and Tank, D. W., ''Computing with Neural Circuits: A Model,'' *Science 233*, 625-633 (1986).

[19] Hopfield, J.J., ''Neural networks and physical systems with emergent collective computational abilities,'' *Proc. Nat. Acad. USA 79*, 2554-2558 (1982).

[20] Feldman, J. A., ''Connectionsn,'' *Byte* 277-284 (April 1987).

[21] Szu, H., ''Globally Connected Network Models for Computing Using Fine-Grained Processing Elements,'' *Proceedings of LASER-85* (1985).

[22] Ingber, L., ''Statistical mechanics of neocortical interactions. EEG dispersion relations,'' *IEEE Trans. Biomed. Eng. 32*, 91-94 (1985).

[23] Ingber, L., ''Statistical mechanics of neocortical interactions. Dynamics of synaptic modification,'' *Phys. Rev. A 28*, 395-416 (1983).

[24] Ingber, L., ''Statistical mechanics of neocortical interactions: Stability and duration of the 7±2 rule of short-term-memory capacity,'' *Phys. Rev. A 31*, 1183-1186 (1985).

[25] Ingber, L., ''Towards a unified brain theory,'' *J. Social Biol. Struct. 4*, 211-224 (1981).

[26] Haken, H., *Synergetics*, 3rd ed. (Springer, New York, 1983).

[27] Peretto, P. and Niez, J., ''Stochastic Dynamics of Neural Networks,'' *IEEE Trans. Syst. Man Cyber. SMC-16*, 73-83 (1986).

[28] Nunez, P. L., *Electric Fields in the Brain: The Neurophysics of EEG* (Oxford University, New York, 1981).

[29] Fishman, G. S., *Concepts and Methods in Discrete Event Digital Simulation* (Wiley, New York, 1973).

[30] Wehner, M. F. and Wolfer, W. G., ''Numerical evaluation of path-integral solutions to Fokker-Plank equations,'' *Physical Review A 27*, 2663-2670 (1983).

[31] Wehner, M. F. and Wolfer, W. G., ''Numerical evaluation of path-integral solutions to Fokker-Plank equations. III. Time and functionally dependent coefficients,'' *Physical Review A 35*, 1795-1801 (1987).

[32] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., ''Equation of State Calculations by Fast Computing Machines,'' *The Journal of Chemical Physics 21*, 1087-1092 (1953).

[33] Landau, D. and Alben, R., ''Monte Carlo Calculations as an Aid in Teaching Statistical Mechanics,'' *AJP 41*, 394-400 (1973).

[34] Port, O., ''Computers that Come Awfully Close to Thinking,'' *Business Week* 92-96 (2 June 1987).

[35] Connell, J., Ingber, L., and Yost, C., ''A Statistical Mechanical Virtual Neural Computer,'' in *1987 Symposium on C3 Research*, ed. by M. Sovereign (National Defense University, Washington, DC, 1987).

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Professor John F. Powers Chairman Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5004	1
4.	Professor Lester Ingber Physics Department Naval Postgraduate School Monterey, CA 93943-5004	30
5.	Professor Carl Jones Chairman C3 Academic Group Naval Postgraduate School Monterey, CA 93943-5004	2
6.	Professor Kai E. Woehler Chairman, Physics Department Naval Postgraduate School Monterey, CA 93943-5004	2
7.	Dean Gordon Schacher Dean of Science and Engineering Naval Postgraduate School Monterey, CA 93943-5004	1
8.	Professor Don Harrison Physics Department Naval Postgraduate School Monterey, CA 93943-5004	1

9.	Maj Richard Adams USAF Naval Postgraduate School Monterey, CA 93943-5004	1
10.	MAJ Bernie Galing USA TRADOC Monterey Monterey, CA 93943-5004	1
11.	Dr. S. Christian Simonson III Conflict Simulation Center Evaluation and Planning Division Lawrence Livermore National Laboratory Livermore, CA 94550	1
12.	Dr. J. B. Adams Sandia Laboratories Livermore, CA 94550	1
13.	COL Don Blumenthal USA (ret) Conflict Simulation Center Evaluation and Planning Division Lawrence Livermore National Laboratory Livermore, CA 94550	1
14.	Dr. William G. Wolfer Sandia Laboratories Livermore, CA 94550	1
15.	Dr. Mike F. Wehner Lawrence Livermore National Laboratory Livermore, CA 94550	1
16.	Capt Stephen C. Upton USMC Plans D Division MCDEC Quantico, VA 22134-5080	1
17.	Adrian S. Yano 2124 Kittredge Street Berkeley, CA 94704	1

18. Professor Aram Mekjian — 1
The State University of New Jersey
Rutgers
Department of Physics and Astronomy
Piscataway, NJ 08854

19. Professor Andrew U. Meyer — 1
New Jersey Institute of Technology
Newark, NJ 07102

20. LCDR Charles P. Yost USN — 5
Fleet Logistics Support Squadron Twenty-Two
FPO New York, NY 09540

21. Professor Vincent Y. Lum — 2
Chairman Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5004

22. CDR John Connell USN — 1
8010 Orange Plank Dr.
Springfield, VA 22153

23. Mike Mudurian — 1
Code 421
Naval Ocean Systems Center (NOSC)
San Diego, CA 92152

24. Michael Melich — 1
Naval Research Laboratory
Code 7570
Washington, DC 20375

25. COL Gary Q. Coe, USA — 1
Chief Modeling & Analysis Division (J6-F)
Office of the Joint Chiefs of Staff
The Pentagon, Room 1D827
Washington, DC 20301-5000

26. Professor Michael Sovereign — 1
USRADCO
APO New York, NY 09159

27.	Dr. Harold Szu Code 5709 Naval Research Lab Washington, DC 20375-5000	1
28.	CPT Thomas Moore, USA Y Division UC Lawrence Livermore Lab Livermore, CA 94550	1
29.	Professor Bruce MacLennan Department of Computer Science Ayers Hall University of Tennessee Knoxville, TN 37996-1301	1

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

Thesis
Y62
c.1

Yost
A virtual statistical mechanical neural computer.

Thesis
Y62
c.1

Yost
A virtual statistical mechanical neural computer.

AMERICAN BINDERY
TOPEKA KANSAS

thesY62
A virtual statistical mechanical nueral
3 2768 000 78757 6
DUDLEY KNOX LIBRARY